```
/*************************************************************************
 * srv.c
 * License server server program
 */
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <signal.h>
#include <sys/errno.h>

#define SERVER_PORTNUM  2020            /* Our server's port number */
#define MSGLEN          128             /* Size of our datagrams */
#define TICKET_AVAIL    -1              /* Slot is available for use */
#define EXPIRE_INTERVAL 60              /* Expire every 60 seconds */
#define MAXUSERS        3               /* Only 3 users for us */
#define REPORT_PREFIX   "\t\t\tSRV:"    /* makes narration easier to read */
#define SYSERROR(x)     { perror(x); exit(-1); }

/*************************************************************************
 * Important variables
 */
static int ticket_array[MAXUSERS];      /* Our ticket array */
static int sd = -1;                     /* Our socket */
static int num_tickets_out = 0;         /* Number of tickets outstanding */
extern int errno;

/*************************************************************************
 * shut_down
 * Shut down cleanly when we get a signal.
 * Results: NEVER RETURNS
 */
void shut_down()
{
        /* Close socket and exit */
        (void) shutdown(sd, 2);
        (void) close(sd);
        exit(0);
} /* shut_down */

/*************************************************************************
 * ticket_expire
 * go through all tickets and get rid of any dead ones
 * Results: none
 */
void ticket_expire()
{
        int x;

        /* Go through the array and look for tickets that belong to
         * processes that are dead.
         */
        for(x = 0; x < MAXUSERS; x++) {
                if((ticket_array[x] != TICKET_AVAIL) &&
                   (kill(ticket_array[x], 0) == -1) && (errno == ESRCH)) {
                        /* Process is gone - free up slot */
                        fprintf(stderr,
                                "%s Process %d gone - freeing ticket\n",
                                        REPORT_PREFIX, ticket_array[x]);
                        ticket_array[x] = TICKET_AVAIL;
                        num_tickets_out--;
                }
        }
        /* Now reset the alarm clock */
        (void) alarm(EXPIRE_INTERVAL);
        return;
} /* ticket_expire */
```

```
/*****************************************************************************
 * do_hello
 * Give out a ticket if any are available
 * IN  msg_p                    message received from client
 * Results: ptr to response
 * NOTE: return is in static buffer overwritten by each call.
 */
static char *do_hello(msg_p)
char *msg_p;
{
        int x;
        static char replybuf[MSGLEN];

        /* Got a ticket request - see if we can give out a ticket. */
        if(num_tickets_out >= MAXUSERS) {
                /* No tickets available */
                return("FAIL no tickets available");
        } else {
                /* Give out a ticket */
                for(x = 0; x < MAXUSERS; x++)
                        if(ticket_array[x] == TICKET_AVAIL)
                                break;

                /* Just a sanity check - should never happen */
                if(x == MAXUSERS) {
                        fprintf(stderr, "%s database corrupt\n", REPORT_PREFIX);
                        return("FAIL database corrupt");
                } else {
                        /* Got it OK! We'll make a ticket for this process
                         * and keep track of the PID (their "name") and
                         * the slot it's in (the "dongle number").
                         *
                         * Thus, our ticket string looks like:
                         *      pid.slot
                         */
                        ticket_array[x] = atoi(msg_p + 5); /* get pid in msg */
                        sprintf(replybuf, "TICK %d.%d", ticket_array[x], x);
                        num_tickets_out++;
                }
        }
        /* Return our response */
        return(replybuf);

} /* do_hello */
/*****************************************************************************
 * do_goodbye
 * Take back ticket client is returning
 * IN  msg_p                    message received from client
 * Results: ptr to response
 * NOTE: return is in static buffer overwritten by each call.
 */
static char *do_goodbye(msg_p)
char *msg_p;
{
        int pid, slot;                    /* ticket components */

        /* The user's giving us back a ticket.  First we need to get
         * the ticket out of the message, which looks like:
         *
         *      GBYE pid.slot
         */
        if((sscanf((msg_p + 5), "%d.%d", &pid, &slot) != 2) ||
           (ticket_array[slot] != pid)) {
                fprintf(stderr, "%s Bogus ticket \"%s\"\n", msg_p + 5,
                                REPORT_PREFIX);
                return("FAIL invalid ticket");
        }
        /* The ticket is valid.  Release it. */
        ticket_array[slot] = TICKET_AVAIL;
        num_tickets_out--;

        /* Return response */
        return("TATA See ya!");

} /* do_goodbye */
```

```
/*****************************************************************************
 * do_validate
 * Validate client's ticket
 * IN  msg_p                    message received from client
 * Results: ptr to response
 * NOTE: return is in static buffer overwritten by each call.
 */
static char *do_validate(msg_p)
char *msg_p;
{
        int pid, slot;              /* components of ticket */

        /* The user's giving us a ticket to validate.  First we need to get
         * the ticket out of the message, which looks like:
         *
         *      VALD pid.slot
         */
        if((sscanf((msg_p + 5), "%d.%d", &pid, &slot) != 2) ||
           (ticket_array[slot] != pid)) {
                fprintf(stderr, "%s Bogus ticket \"%s\"\n",
                                REPORT_PREFIX, msg_p + 5);
                return("FAIL invalid ticket");
        }
        /* If we got here the ticket's good */
        return("GOOD Valid ticket");
} /* do_validate */

/*****************************************************************************
 * main
 * main processing loop for server program
 * Results: none
 */
main()
{
        int pid;
        char buf[MSGLEN];
        int secs_left = EXPIRE_INTERVAL;
        struct sockaddr_in server_addr;
        struct sockaddr_in client_addr;
        int addrlen;
        char *resp_p;
        int x;
        char *inet_ntoa();

        struct  sockaddr_in   saddr;   /* build our address here */
        struct  hostent         *hp;   /* this is part of our    */
        char    hostname[256];         /* address                */
        int     slen,sock_id,sock_fd;  /* line id, file desc     */
        FILE    *sock_fp;              /* use socket as stream   */
        char    *ctime();              /* convert secs to string */
        long    time(), thetime;       /* time and the val       */

        gethostname( hostname );                /* where am I ?         */
        hp = gethostbyname( hostname );         /* get info about host  */
        bzero( &saddr, sizeof(saddr) );         /* zero struct          */
                                                /* fill in hostaddr     */
        bcopy( hp->h_addr, &saddr.sin_addr, hp->h_length);
        saddr.sin_family = AF_INET ;            /* fill in socket type  */
        saddr.sin_port = htons(SERVER_PORTNUM); /* fill in socket port  */

       /*
        *      step 2: ask kernel for a socket, then bind address
        */

        sd = socket( AF_INET, SOCK_DGRAM, 0 );    /* get a socket */
        if ( sd == -1 )
                SYSERROR("SERVER: socket");

        if ( bind(sd, &saddr, sizeof(saddr)) != 0 )/* bind it to   */
                SYSERROR("SERVER: bind");
```

```
        /* Set up signal handler to clean up on exit */
        (void) signal(SIGINT, shut_down);
        (void) signal(SIGTERM, shut_down);

#ifdef EXPIRE_TICKETS
        /* Set up the alarm clock */
        (void) signal(SIGALRM, ticket_expire);
        alarm(EXPIRE_INTERVAL);
#endif /* EXPIRE_TICKETS */

        /* Initialize ticket database */
        for(x = 0; x < MAXUSERS; x++)
                ticket_array[x] = TICKET_AVAIL;

        /* Set up a loop, listening to client requests forever */
        while(1) {
                /* Get a request */
                addrlen = sizeof(client_addr);
                if(recvfrom(sd, buf, MSGLEN, 0, &client_addr, &addrlen) == -1) {
                        /* If we get an error we'll acknowledge it, but we
                         * must continue so that we can help other clients.
                         *
                         * We ignore EINTR; that's just the alarm handler
                         */
                        if(errno != EINTR)
                                perror("SERVER: recvfrom");
                        continue;
                }
                fprintf(stderr, "%s Got \"%s\" from %s,%d\n",
                        REPORT_PREFIX, buf,
                       inet_ntoa(client_addr.sin_addr), client_addr.sin_port);

#ifdef EXPIRE_TICKETS
                /* Turn off the alarm clock so we're not interrupted in
                 * the middle of things.
                 */
                secs_left = alarm(0);
#endif /* EXPIRE_TICKETS */

                /* Is this a request we know how to deal with?  If so,
                 * do so.  If not, send a failure message back.
                 * We recognize the following:
                 *      HELO client-id-string
                 *      GBYE ticket-id-string
                 *      VALD ticket-id-string
                 */
                if(strncmp(buf, "HELO", 4) == 0)
                        resp_p = do_hello(buf);
                else if(strncmp(buf, "GBYE", 4) == 0)
                        resp_p = do_goodbye(buf);
                else if(strncmp(buf, "VALD", 4) == 0)
                        resp_p = do_validate(buf);
                else
                        resp_p = "FAIL Invalid Request";

                /* Now send the response */
                fprintf(stderr, "%s sending response \"%s\"\n",
                                REPORT_PREFIX, resp_p);

                if(sendto(sd, resp_p, MSGLEN, 0, &client_addr, addrlen) == -1) {
                        /* If we get an error we'll acknowledge it, but we
                         * must continue so that we can help other clients.
                         */
                        perror("SERVER: sendto");
                        continue;
                }
#ifdef EXPIRE_TICKETS
                /* Turn the alarm clock back on since we're idle again */
                if(secs_left == 0)
                        secs_left = EXPIRE_INTERVAL;    /* Reset timer */
                (void) alarm(secs_left);
#endif /* EXPIRE_TICKETS */
                /* Just go back and get the next msg! */
        }
        /* NOTREACHED */
} /* main */
```

```
/*************************************************************************
 * clnt.c
 * License server client
 */

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#define SERVER_PORTNUM  2020            /* Our server's port number */
#define MSGLEN          128             /* Size of our datagrams */
#define oops(p) { perror(p); exit(1) ; }

/*************************************************************************
 * Important variables used throughout
 */
static int pid = -1;                    /* Our PID */
static int sd = -1;                     /* Our communications socket */
static struct sockaddr_in server_addr;  /* Server address */
static char ticket_buf[128];            /* Buffer to hold our ticket */
static have_ticket = 0;                 /* Set when we have a ticket */

/*************************************************************************
 * do_transaction
 * Send a request to the server and get a response back
 * IN  sd               socket descriptor
 * IN  msg_p            message to send
 * IN  addr_p           server's address
 * Results: pointer to message string, or NULL for error
 *                      NOTE: pointer returned is to static storage
 *                      overwritten by each successive call.
 */
static char *do_transaction(sd, msg_p, addr_p)
int sd;
char *msg_p;
struct sockaddr_in *addr_p;
{
        static char buf[MSGLEN];

        struct sockaddr_in retaddr;
        int addrlen;
        int ret;

        /* First send the message */
        strncpy(buf, msg_p, MSGLEN);
        if((ret = sendto(sd, buf, MSGLEN, 0, addr_p,
            sizeof(struct sockaddr_in))) == -1) {
                sprintf(buf, "CLIENT [%d]: sendto", pid);
                perror(buf);
                return((char *) 0);
        }

        /* Get the response back */
        if((ret = recvfrom(sd, buf, MSGLEN, 0, &retaddr, &addrlen)) == -1) {
                sprintf(buf, "CLIENT [%d]: recvfrom", pid);
                perror(buf);
                return((char *) 0);
        }

        /* Now return the message itself */
        return(buf);
} /* do_transaction */
```

```
/*****************************************************************************
 * get_ticket
 * get a ticket from the license server
 * Results: 0 for success, 1 for failure
 */
static int get_ticket()
{
        char *resp_p;
        char buf[MSGLEN];

        /* If we already have a ticket, don't ask for another */
        if(have_ticket)
                return(0);

        /* Now set up a request to send to the server.  We use our
         * PID to identify ourselves to the server.
         */
        sprintf(buf, "HELO %d", pid);

        /* Perform the transaction itself */
        if((resp_p = do_transaction(sd, buf, &server_addr)) == (char *) 0)
                return(-1);

        /* Now parse the response and see if we got a ticket.  If we did,
         * the message will be:
         *      TICK ticket-string
         * If not, it will be:
         *      FAIL failure-msg
         */
        if(strncmp(resp_p, "TICK", 4) != 0) {
                if(strncmp(resp_p, "FAIL") != 0) {
                        fprintf(stderr, "CLIENT [%d]: unknown msg \"%s\"\n",
                                pid, resp_p);
                        return(-1);
                } else {
                        fprintf(stderr, "CLIENT [%d]: couldn\'t get ticket.\n",
                                pid);
                        return(-1);
                }
        } else
                fprintf(stderr, "CLIENT [%d]: got ticket!\n", pid);

        /* Save the ticket string */
        strcpy(ticket_buf, resp_p + 5); /* ticket-string after "TICK " */
        have_ticket = 1;                /* We have a ticket */
        return(0);
} /* get_ticket */

/*****************************************************************************
 * release_ticket
 * Give a ticket back to the server
 * Results: 0 for success, -1 for failure
 */
static int release_ticket()
{
        char buf[MSGLEN];
        char *resp_p;

        /* If we don't have one, there's none to give back */
        if(!have_ticket)
                return(0);

        /* Now we'll give it back - format the message */
        sprintf(buf, "GBYE %s", ticket_buf);

        /* Perform the transaction itself */
        if((resp_p = do_transaction(sd, buf, &server_addr)) == (char *) 0)
                return(-1);
```

```
        /* Now see what the server sent us.  If everything went OK, we'll
         * get a message like:
         *      TATA info-string
         * If it failed, we'll get a message like:
         *      FAIL error-string
         * Otherwise, something we don't understand went wrong.
         */
        if(strncmp(resp_p, "TATA", 4) == 0) {
                fprintf(stderr, "CLIENT [%d]: released ticket OK\n", pid);
                return(0);
        } else if(strncmp(resp_p, "FAIL", 4) == 0)
                fprintf(stderr, "CLIENT [%d]: release failed \"%s\"\n",
                        pid, resp_p + 5);
        else
                fprintf(stderr, "CLIENT [%d]: unknown response \"%s\"\n",
                        pid, resp_p);

        /* Something went wrong */
        return(-1);
} /* release_ticket */

/*************************************************************************
 * validate_ticket
 * Make sure the ticket we have is still good
 * Results: 0 for success, -1 for failure
 */
static int validate_ticket()
{
        char buf[MSGLEN];
        char *resp_p;

        /* If we don't have one, there's none to validate */
        if(!have_ticket)
                return(-1);

        /* If we're not validating, we skip the test, and just always say
         * it's OK
         */
#ifdef VALIDATE_TICKETS
        sprintf(buf, "VALD %s", ticket_buf);

        /* Send the message - don't exit if this fails, since we must
         * free the ticket!
         */

        /* Perform the transaction itself - don't exit if we can't reach
         * the server, though, as we still have to free the ticket.
         */
        if((resp_p = do_transaction(sd, buf, &server_addr)) == (char *) 0) {
                have_ticket = 0;
                return(-1);
        }

        /* Now make sure we got a valid response.  We should either get
         * back a validation:
         *      GOOD response-string
         * or a failure:
         *      FAIL error-string
         */
        if(strncmp(resp_p, "FAIL", 4) == 0) {
                /* We got a failure msg - the ticket's no good, so exit. */
                fprintf(stderr, "CLIENT [%d]: validation failed - \"%s\"\n",
                        pid, buf + 5);
                have_ticket = 0;
                return(-1);
        } else if(strncmp(resp_p, "GOOD", 4) != 0) {
                /* We don't know what we got - get out of here */
                fprintf(stderr, "CLIENT [%d]: unknown msg - \"%s\"\n",
                        pid, buf + 5);
                have_ticket = 0;
                return(-1);
        }
#endif /* VALIDATE_TICKETS */

        /* we validated OK */
        return(0);
} /* validate_ticket */
```

```
/*****************************************************************************
 * main
 * main processing loop for client program
 * Results: none
 */
main()
{
        char buf[MSGLEN];
        struct hostent      *hp;            /* used to get number */
        char   message[BUFSIZ];             /* to receive message */
        int    messlen;                     /* for message length */
        char   hostname[256];

       /*
        *      build the network address of where we want to call
        */

        gethostname( hostname );
        hp = gethostbyname( hostname );
        if ( hp == NULL ) oops("no such computer");

        bzero( &server_addr, sizeof( server_addr ) );   /* zero the address     */
        server_addr.sin_family = AF_INET ;              /* fill in socket type  */
                                                       /* and machine address  */
        bcopy( hp->h_addr, &server_addr.sin_addr, hp->h_length);
        server_addr.sin_port = htons(SERVER_PORTNUM);   /* host to num short    */


        /* Set our PID so we can use it in msgs, etc. later */
        pid = getpid();

        /* Set up a datagram socket. */
        if((sd = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
                perror("CLIENT: socket");
                exit(-1);
        }

        /* Try to get a ticket - get_ticket() prints a message if this
         * part fails.
         */
        if(get_ticket())
                exit(-1);

        /* If we got here OK, we got a ticket.  Let's just go to sleep for
         * a few seconds, and then we'll release it.
         */
        fprintf(stderr, "CLIENT [%d]: sleeping...\n", pid);
        sleep(10);

#ifdef  VALIDATE_TICKETS
        /* Now let's try and validate the ticket */
        if(validate_ticket())                           /* Version 3 only */
                exit(-1);

        /* If we got here, we validated OK. */
        fprintf(stderr,
                "CLIENT [%d]: validated ticket - sleeping some more ...\n",
                pid);
        sleep(5);
#endif /* VALIDATE_TICKETS */

        /* Release the ticket */
        (void) release_ticket();

        /* Now we're done.  Let's just clean up and get out of here */
        fprintf(stderr, "CLIENT [%d]: released ticket.  exiting...\n", pid);
        (void) close(sd);
        exit(0);
} /* main */
```