```c
#if defined(macintosh) || defined(__MWERKS__)

#include <limits.h>
#include "uio.h"

#include<stdlib.h>
#define h_errno 0
typedef unsigned int uint;
#define NO_THREADS

#else

#include <stdio.h>
#include <errno.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/socket.h>

#endif

#include "gss.h"

#define TIMEOUT 10
```

```
/*****************************************************************************
 *
 * connect_socket()
 *
 * open a non-blocking socket and connect
 *
 *****************************************************************************/

char *connect_socket(int *psock, struct sockaddr_in *paddr)
{
  fd_set                fdset;
  struct timeval        timeout = {TIMEOUT, 0};
  int                   sock = -1;
  char                  *text = NULL;

  /* open socket */
  sock = socket(AF_INET, SOCK_STREAM, IPPROTO_IP);
  if (sock < 0) {
    text = unix_errmsg("while creating socket", errno);
    goto Cleanup;
  }

  /* make socket non-blocking */
  if (fcntl(sock, F_SETFL, O_NDELAY|fcntl(sock, F_GETFL))) {
    text = unix_errmsg("while making socket non-blocking", errno);
    goto Cleanup;
  }

  /* connect */
  if (connect(sock, (struct sockaddr *)paddr, sizeof(*paddr))
      && errno && errno != EINPROGRESS) {
    text = unix_errmsg("while connecting", errno);
    goto Cleanup;
  }

  /* wait until the socket is ready for write */
  FD_ZERO(&fdset);
  FD_SET(sock, &fdset);
  if (select(sock+1, NULL, &fdset, NULL, &timeout) != 1) {
    text = unix_errmsg("while selecting", errno);
    goto Cleanup;
  }

 Cleanup:
  if (text && sock >=0) {
    close(sock);
    sock = -1;
  }
  *psock = sock;
  return text;
}
```

```
/*****************************************************************************
 *
 * send_token()
 *
 * send tokens
 *
 ****************************************************************************/

int send_token(int sock, gss_buffer_desc *tok)
{
  struct iovec          iov[2];
  long                  len;
  struct timeval        timeout = {TIMEOUT, 0};
  fd_set                fdset;
  int                   n;

  FD_ZERO(&fdset);
  FD_SET(sock, &fdset);

  iov[0].iov_base = (char*)&len;
  iov[0].iov_len  = sizeof(len);
  iov[1].iov_base = (char*)tok->value;
  iov[1].iov_len  = tok->length;

  len = htonl(tok->length);

  if (select(sock+1, NULL, &fdset, NULL, &timeout) != 1) goto Die;
  n = writev(sock, iov, 2);
  if (n < sizeof(len)) goto Die;
  len = n - sizeof(len);

  for(; len<tok->length; len+=n) {
    if (select(sock+1, NULL, &fdset, NULL, &timeout) != 1) goto Die;
    if ((n=write(sock, ((char*)tok->value)+len, tok->length-len)) <= 0)
      goto Die;
  }

  return 0;

 Die:
  if (!errno) errno = ECONNABORTED;
  return -1;
}
```

```
/******************************************************************************
 *
 * recv_token()
 *
 * receive tokens
 *
 ******************************************************************************/

int recv_token(int sock, gss_buffer_desc *tok)
{
  char                  buf[8196];
  struct iovec          iov[2];
  long                  len;
  int                   n;
  struct timeval        timeout = {TIMEOUT, 0};
  fd_set                fdset;

  FD_ZERO(&fdset);
  FD_SET(sock, &fdset);

  iov[0].iov_base = (char*)&len;
  iov[0].iov_len  = sizeof(len);
  iov[1].iov_base = buf;
  iov[1].iov_len  = sizeof(buf);

  tok->length = 0;

  if (select(sock+1, &fdset, NULL, NULL, &timeout) != 1) goto Die;
  n = readv(sock, iov, 2);
  if (n < sizeof(len)) goto Die;

  tok->length = ntohl(len);
  tok->value = (char*)malloc(tok->length);
  if (!tok->value) goto Die;
  if (len=n-sizeof(len))
    memcpy(tok->value, buf, len);

  for(; len<tok->length; len+=n) {
    if (select(sock+1, &fdset, NULL, NULL, &timeout) != 1) goto Die;
    if ((n=read(sock, ((char*)tok->value)+len, tok->length-len)) <= 0)
      goto Die;
  }

  return 0;

 Die:
  n = errno ? errno : ECONNABORTED;
  if (tok->length) free(tok->value);
  tok->length = 0;
  errno = n;
  return -1;
}
```

```
/******************************************************************************
 *
 * unix_errmsg(), gss_errmsg()
 *
 * format error messages as char*
 *
 ******************************************************************************/

char *unix_errmsg(char *s, int err)
{
  char *buf = malloc(1024);
  sprintf(buf, "%s: %s", s, strerror(err));
  return buf;
}

static
void gss_errmsg1(OM_uint32 stat, int code, gss_buffer_t errmsg)
{
  OM_uint32             ignore;
  uint                  err_ctx = 0;
  int                   buflen = strlen((char*)(errmsg->value));
  gss_buffer_desc       line = GSS_C_EMPTY_BUFFER;

  if (stat == GSS_S_COMPLETE) return;

  do {
    (void)gss_display_status(&ignore, stat, code, GSS_C_NULL_OID, &err_ctx, &lin
e);

    if (line.length + 3 + buflen >= errmsg->length)
      errmsg->value = realloc(errmsg->value, errmsg->length += line.length + 3);

    sprintf((char*)(errmsg->value) + buflen, "%.*s. ", line.length, line.value);

    buflen += line.length + 2;

    (void)gss_release_buffer(&ignore, &line);

  } while(err_ctx);
}

char *gss_errmsg(char *s, OM_uint32 maj_stat, OM_uint32 min_stat)
{
  gss_buffer_desc       errmsg;

  errmsg.length = 256;
  errmsg.value = malloc(errmsg.length);

  sprintf(errmsg.value, "GSS-API error %s: ", s);
  gss_errmsg1(maj_stat, GSS_C_GSS_CODE, &errmsg);
  gss_errmsg1(min_stat, GSS_C_MECH_CODE, &errmsg);

  return errmsg.value;
}
```

```
/******************************************************************************
 *
 * server_acquire_creds()
 *
 * acquire GSS server credentials
 *
 ******************************************************************************/

char *server_acquire_creds(char *service_name, char *mechanism_name, int cred_us
                                              age, gss_cred_id_t *server_creds)
{
  gss_buffer_desc       name_buf;
  OM_uint32             maj_stat, min_stat, ignore;

  gss_name_t            server_name = NULL;
  gss_OID_set           sec_mechs = GSS_C_NO_OID_SET;
  gss_OID               doid = GSS_C_NO_OID;
  gss_buffer_desc       fullname_buf = GSS_C_EMPTY_BUFFER;
  char                  *text = NULL;

  *server_creds =  GSS_C_NO_CREDENTIAL;

  name_buf.value = service_name;
  name_buf.length = strlen(name_buf.value) + 1;
  maj_stat = gss_import_name(&min_stat, &name_buf,
                             (gss_OID)gss_nt_service_name, &server_name);
  if (maj_stat != GSS_S_COMPLETE) {
    text = gss_errmsg("importing name", maj_stat, min_stat);
    goto Cleanup;
  }

  name_buf.value = mechanism_name;
  name_buf.length = strlen(name_buf.value) + 1;
  maj_stat = gss_str_to_oid(&min_stat, &name_buf, &doid);
  if (maj_stat != GSS_S_COMPLETE) {
    text = gss_errmsg("parsing security oid", maj_stat, min_stat);
    goto Cleanup;
  }

  maj_stat = gss_create_empty_oid_set(&min_stat, &sec_mechs);
  if (maj_stat != GSS_S_COMPLETE) {
    text = gss_errmsg("creating empty oid set", maj_stat, min_stat);
    goto Cleanup;
  }

  maj_stat = gss_add_oid_set_member(&min_stat, doid, &sec_mechs);
  if (maj_stat != GSS_S_COMPLETE) {
    text = gss_errmsg("adding oid set member", maj_stat, min_stat);
    goto Cleanup;
  }
```

```
  maj_stat = gss_acquire_cred(&min_stat, server_name, 0,
                              sec_mechs, cred_usage,
                              server_creds, NULL, NULL);
  if (maj_stat != GSS_S_COMPLETE) {
    text = gss_errmsg("acquiring credentials", maj_stat, min_stat);
    goto Cleanup;
  }

  (void) gss_display_name(&ignore, server_name, &fullname_buf, &doid);
  if (fullname_buf.length > 0) {
    memcpy(service_name, fullname_buf.value, fullname_buf.length);
    service_name[fullname_buf.length] = '\0';
  }

 Cleanup:
  (void)gss_release_name(&ignore, &server_name);
  (void)gss_release_buffer(&ignore, &fullname_buf);
  (void)gss_release_oid_set(&ignore, &sec_mechs);
  (void)gss_release_oid(&ignore, &doid);
  return text;
}
```

```
/*****************************************************************************
 *
 * server_establish_context()
 *
 * establish server GSS context
 *
 *****************************************************************************/

char *server_establish_context(int sock, gss_cred_id_t server_creds,
                               gss_ctx_id_t *context, gss_buffer_t client_name)
{
  OM_uint32              maj_stat, min_stat, ignore;
  OM_uint32              ret_flags;

  gss_name_t             client = NULL;
  gss_OID                doid = GSS_C_NO_OID;
  gss_buffer_desc        send_tok = GSS_C_EMPTY_BUFFER;
  gss_buffer_desc        recv_tok = GSS_C_EMPTY_BUFFER;
  char                   *errmsg = NULL;

  *context = GSS_C_NO_CONTEXT;

  do {

    if (recv_token(sock, &recv_tok) < 0) {
      errmsg = unix_errmsg("while receiving", errno);
      goto Cleanup;
    }

    maj_stat =
      gss_accept_sec_context(&min_stat,
                             context,
                             server_creds,
                             &recv_tok,
                             GSS_C_NO_CHANNEL_BINDINGS,
                             &client,
                             &doid,
                             &send_tok,
                             &ret_flags,
                             NULL,      /* ignore time_rec */
                             NULL);     /* ignore del_cred_handle */
    (void)gss_release_buffer(&ignore, &recv_tok);

    if (maj_stat!=GSS_S_COMPLETE && maj_stat!=GSS_S_CONTINUE_NEEDED) {
      errmsg = gss_errmsg("accepting context", maj_stat, min_stat);
      goto Cleanup;
    }
```

```
    if (send_tok.length != 0) {
      if (send_token(sock, &send_tok) < 0) {
        errmsg = unix_errmsg("while sending", errno);
        goto Cleanup;
      }
      (void)gss_release_buffer(&ignore, &send_tok);
    }

  } while (maj_stat == GSS_S_CONTINUE_NEEDED);

  maj_stat = gss_display_name(&min_stat, client, client_name, &doid);
  if (maj_stat != GSS_S_COMPLETE) {
    errmsg = gss_errmsg("displaying name", maj_stat, min_stat);
    goto Cleanup;
  }

 Cleanup:
  (void)gss_release_name(&ignore, &client);
  (void)gss_release_buffer(&ignore, &send_tok);
  (void)gss_release_buffer(&ignore, &recv_tok);
  (void)gss_release_oid(&ignore, &doid);
  if (errmsg && *context != GSS_C_NO_CONTEXT) {
    (void)gss_delete_sec_context(&ignore, context, &send_tok);
    (void)gss_release_buffer(&ignore, &send_tok);
  }

  return errmsg;
}
```

```
/******************************************************************************
 *
 * client_establish_context()
 *
 * establish client GSS context
 *
 ******************************************************************************/

char *client_establish_context(int s, gss_cred_id_t client_creds,
                             char *service_name, char *mechanism_name, gss_ctx_i
                                                                d_t *gss_context)
{
  gss_buffer_desc       name_buf;
  gss_buffer_t          token_ptr;
  OM_uint32             maj_stat, min_stat, ignore;

  gss_buffer_desc       send_tok = GSS_C_EMPTY_BUFFER;
  gss_buffer_desc       recv_tok = GSS_C_EMPTY_BUFFER;
  gss_name_t            target_name = NULL;
  gss_OID               secoid = GSS_C_NO_OID;
  char                  *errmsg = NULL;

  token_ptr = GSS_C_NO_BUFFER;
  *gss_context = GSS_C_NO_CONTEXT;

  name_buf.value = service_name;
  name_buf.length = strlen(service_name) + 1;
  maj_stat = gss_import_name(&min_stat, &name_buf,
                             (gss_OID)gss_nt_service_name, &target_name);
  if (maj_stat != GSS_S_COMPLETE) {
    errmsg = gss_errmsg("parsing name", maj_stat, min_stat);
    goto Cleanup;
  }

  name_buf.value = mechanism_name;
  name_buf.length = strlen(mechanism_name) + 1;
  maj_stat = gss_str_to_oid(&min_stat, &name_buf, &secoid);
  if (maj_stat != GSS_S_COMPLETE) {
    errmsg = gss_errmsg("parsing security oid", maj_stat, min_stat);
    goto Cleanup;
  }

  for(;;) {
    maj_stat =
      gss_init_sec_context(&min_stat,
                           client_creds,
                           gss_context,
                           target_name,
                           secoid,
                           GSS_C_MUTUAL_FLAG | GSS_C_REPLAY_FLAG,
                           0,
                           NULL,        /* no channel bindings */
                           token_ptr,
                           NULL,        /* ignore mech type */
                           &send_tok,
                           NULL,        /* ignore ret_flags */
                           NULL);       /* ignore time_rec */

    if (token_ptr != GSS_C_NO_BUFFER)
      (void) gss_release_buffer(&ignore, &recv_tok);
```

```
    if (maj_stat!=GSS_S_COMPLETE && maj_stat!=GSS_S_CONTINUE_NEEDED) {
      errmsg = gss_errmsg("initializing context", maj_stat, min_stat);
      goto Cleanup;
    }

    if (send_tok.length && send_token(s, &send_tok) < 0) {
      errmsg = unix_errmsg("while sending", errno);
      goto Cleanup;
    }
    (void)gss_release_buffer(&ignore, &send_tok);

    if (maj_stat != GSS_S_CONTINUE_NEEDED) break;

    if (recv_token(s, &recv_tok) < 0) {
      errmsg = unix_errmsg("while receiving", errno);
      goto Cleanup;
    }

    token_ptr = &recv_tok;
  }

 Cleanup:
  (void)gss_release_buffer(&ignore, &send_tok);
  (void)gss_release_buffer(&ignore, &recv_tok);
  (void)gss_release_name(&ignore, &target_name);
  (void)gss_release_oid(&ignore, &secoid);

  return errmsg;
}
```

```
/******************************************************************************
 *
 * awful, ugly kludge to make GSS thread-safe....
 *
 *      the following should be deleted as soon as GSS is
 *      guaranteed to be thread-safe
 *
 ******************************************************************************/

#if !defined(NO_THREADS)

#include <thr.h>

static MUTEX lock;

#undef gss_accept_sec_context
OM_uint32 INTERFACE TS_gss_accept_sec_context
(OM_uint32 FAR * A,
 gss_ctx_id_t FAR * B,
 gss_cred_id_t C,
 gss_buffer_t D,
 gss_channel_bindings_t E,
 gss_name_t FAR * F,
 gss_OID FAR * G,
 gss_buffer_t H,
 OM_uint32 FAR * I,
 OM_uint32 FAR * J,
 gss_cred_id_t FAR * K
 )
{
  OM_uint32     maj_stat;
  LOCK(&lock);
  maj_stat = gss_accept_sec_context(A, B, C, D, E, F, G, H, I, J, K);
  UNLOCK(&lock);
  return maj_stat;
}

#undef gss_acquire_cred
OM_uint32 INTERFACE TS_gss_acquire_cred
(OM_uint32 FAR * A,
 gss_name_t B,
 OM_uint32 C,
 gss_OID_set D,
 gss_cred_usage_t E,
 gss_cred_id_t FAR * F,
 gss_OID_set FAR * G,
 OM_uint32 FAR * H
 )
{
  OM_uint32     maj_stat;
  LOCK(&lock);
  maj_stat = gss_acquire_cred(A, B, C, D, E, F, G, H);
  UNLOCK(&lock);
  return maj_stat;
}

#undef gss_delete_sec_context
OM_uint32 INTERFACE TS_gss_delete_sec_context
(OM_uint32 FAR * A,
 gss_ctx_id_t FAR * B,
 gss_buffer_t C
 )
{
  OM_uint32     maj_stat;
```

```
  LOCK(&lock);
  maj_stat = gss_delete_sec_context(A, B, C);
  UNLOCK(&lock);
  return maj_stat;
}

#undef gss_display_name
OM_uint32 INTERFACE TS_gss_display_name
(OM_uint32 FAR * A,
 gss_name_t B,
 gss_buffer_t C,
 gss_OID FAR * D
 )
{
  OM_uint32     maj_stat;
  LOCK(&lock);
  maj_stat = gss_display_name(A, B, C, D);
  UNLOCK(&lock);
  return maj_stat;
}

#undef gss_display_status
OM_uint32 INTERFACE TS_gss_display_status
(OM_uint32 FAR * A,
 OM_uint32 B,
 int C,
 gss_OID D,
 OM_uint32 FAR * E,
 gss_buffer_t F
 )
{
  OM_uint32     maj_stat;
  LOCK(&lock);
  maj_stat = gss_display_status(A, B, C, D, E, F);
  UNLOCK(&lock);
  return maj_stat;
}

#undef gss_import_name
OM_uint32 INTERFACE TS_gss_import_name
(OM_uint32 FAR * A,
 gss_buffer_t B,
 gss_OID C,
 gss_name_t FAR * D
 )
{
  OM_uint32     maj_stat;
  LOCK(&lock);
  maj_stat = gss_import_name(A, B, C, D);
  UNLOCK(&lock);
  return maj_stat;
}

#undef gss_init_sec_context
OM_uint32 INTERFACE TS_gss_init_sec_context
(OM_uint32 FAR * A,
 gss_cred_id_t B,
 gss_ctx_id_t FAR * C,
 gss_name_t D,
 gss_OID E,
 OM_uint32 F,
 OM_uint32 G,
 gss_channel_bindings_t H,
 gss_buffer_t I,
 gss_OID FAR * J,
 gss_buffer_t K,
```

```
 OM_uint32 FAR * L,
 OM_uint32 FAR * M
 )
{
  OM_uint32     maj_stat;
  LOCK(&lock);
  maj_stat = gss_init_sec_context(A, B, C, D, E, F, G, H, I, J, K, L, M);
  UNLOCK(&lock);
  return maj_stat;
}

#undef gss_oid_to_str
OM_uint32 INTERFACE TS_gss_oid_to_str
(OM_uint32 FAR * A,
 gss_OID B,
 gss_buffer_t C
 )
{
  OM_uint32     maj_stat;
  LOCK(&lock);
  maj_stat = gss_oid_to_str(A, B, C);
  UNLOCK(&lock);
  return maj_stat;
}

#undef gss_release_buffer
OM_uint32 INTERFACE TS_gss_release_buffer
(OM_uint32 FAR * A,
 gss_buffer_t B
 )
{
  OM_uint32     maj_stat;
  LOCK(&lock);
  maj_stat = gss_release_buffer(A, B);
  UNLOCK(&lock);
  return maj_stat;
}

#undef gss_release_name
OM_uint32 INTERFACE TS_gss_release_name
(OM_uint32 FAR * A,
 gss_name_t FAR * B
 )
{
  OM_uint32     maj_stat;
  LOCK(&lock);
  maj_stat = gss_release_name(A, B);
  UNLOCK(&lock);
  return maj_stat;
}

#undef gss_release_oid_set
OM_uint32 INTERFACE TS_gss_release_oid_set
(OM_uint32 FAR * A,
 gss_OID_set FAR * B
 )
{
  OM_uint32     maj_stat;
  LOCK(&lock);
  maj_stat = gss_release_oid_set(A, B);
  UNLOCK(&lock);
  return maj_stat;
}

#undef gss_seal
OM_uint32 INTERFACE TS_gss_seal
```

```
(OM_uint32 FAR * A,
 gss_ctx_id_t B,
 int C,
 int D,
 gss_buffer_t E,
 int FAR * F,
 gss_buffer_t G
 )
{
  OM_uint32     maj_stat;
  LOCK(&lock);
  maj_stat = gss_seal(A, B, C, D, E, F, G);
  UNLOCK(&lock);
  return maj_stat;
}

#undef gss_str_to_oid
OM_uint32 INTERFACE TS_gss_str_to_oid
(OM_uint32 FAR * A,
 gss_buffer_t B,
 gss_OID FAR * C
 )
{
  OM_uint32     maj_stat;
  LOCK(&lock);
  maj_stat = gss_str_to_oid(A, B, C);
  UNLOCK(&lock);
  return maj_stat;
}

#undef gss_unseal
OM_uint32 INTERFACE TS_gss_unseal
(OM_uint32 FAR * A,
 gss_ctx_id_t B,
 gss_buffer_t C,
 gss_buffer_t D,
 int FAR * E,
 int FAR * F
 )
{
  OM_uint32     maj_stat;
  LOCK(&lock);
  maj_stat = gss_unseal(A, B, C, D, E, F);
  UNLOCK(&lock);
  return maj_stat;
}

#endif
```