LULEÅ UNIVERSITY OF TECHNOLOGY

MALLOC Implementation

LIBMADCAP Manual

Håkan Byström, Maria Isberg, Pia Lindström,
Fredrik Pettersson, Linus Svensson, Åke Östmark

LIBMADCAP Manual
SMD113 − Networking Project

Department of Computer Science and Electrical Engineering
Division of Computer Communications
2001−05−16

Contents

1. Introduction

This manual describes how to write programs which use the libmadcap API in order to allocate multicast addresses from a MADCAP server. The API is only intended for client side use. Note that this is not a complete implementation of all functionality possible for a client program using MADCAP. The functionality included is only the most basic functions.

2. Installation

The library is distributed as a file named `libmadcap-<ver>.tar.gz`, where `<ver>` is the version of the library. The current version is 0.1. In order to extract the files from the archive, use the command

```
tar -xvzf libmadcap-0.1.tar.gz
```

You now have a directory called `libmadcap-0.1`. The rest of the installation procedure is described in the file INSTALLATION in the newly created directory, but if you run as root you should only need to do the following commands.

```
./configure
make
make install
```

These are the installed files:

- `/usr/include/madcap/madcap.h` is the header file containing the declarations of the public functions in the library.
- `/usr/local/lib/libmadcap.a` is the version of the library used for static linking.
- `/usr/local/lib/libmadcap.so` is the version of the library used for dynamic linking.

3. API

This is a listing of the API functions and how to use them.

3.1 Address allocation

There are several functions for allocating addresses. The list of available functions are listed below.

This function has all the possible parameters.

```
int alloc_multicast_addr(uint16 address_family,
                         scope *scope,
                         uint16 min_desired_addresses,
                         uint16 max_desired_addresses,
                         uint32 min_desired_start_time,
                         uint32 max_desired_start_time,
                         uint32 min_desired_lifetime,
                         uint32 max_desired_lifetime,
                         uint32 server,
                         lease *multicast_address_set_lease,
                         int *status);
```

This is the same as the first, except no server has to be provided.

```
int alloc_noserv_addr(uint16 address_family,
                      scope *scope,
                      uint16 min_desired_addresses,
                      uint16 max_desired_addresses,
                      uint32 min_desired_start_time,
                      uint32 max_desired_start_time,
                      uint32 min_desired_lifetime,
                      uint32 max_desired_lifetime,
                      lease *multicast_address_set_lease,
                      int *status);
```

Allocates exactly one address.

```
int alloc_one_addr(uint16 address_family,
                   scope *scope,
                   uint32 min_desired_start_time,
                   uint32 max_desired_start_time,
                   uint32 min_desired_lifetime,
                   uint32 max_desired_lifetime,
                   lease *multicast_address_set_lease,
                   int *status);
```

Allocates addresses for a specific lifetime.

```
int alloc_exact_addr(uint16 address_family,
                     scope *scope,
                     unsigned int min_desired_addresses,
                     unsigned int max_desired_addresses,
                     uint32 min_desired_start_time,
                     uint32 max_desired_start_time,
                     uint32 lifetime,
                     lease *multicast_address_set_lease,
                     int *status);
```

Allocates an address which should start at once.

```
int alloc_immediate_addr(uint16 address_family,
                         scope *scope,
                         int min_desired_addresses,
                         int max_desired_addresses,
                         uint32 min_desired_lifetime,
                         uint32 max_desired_lifetime,
                         lease *multicast_address_set_lease,
                         int *status);
```

The parameters for the functions are:

- `address_family` is a 16 bit unsigned integer identifying the address family from which to allocate the addresses. The implementation currently only supports IPv4 as indicated by MC_IPV4.
- `scope` is the multicast scope where the allocated addresses should be valid. The scope struct is defined as

```
typedef struct scope{
    address_range addresses; /* The set of addresses in the scope. */
    name *name_set;          /* The names for the scope. */
    unsigned int num_names;  /* The number of names for the scope. */
    unsigned int ttl;        /* Default TTL. */
} scope;

typedef struct address_range{
    uint32 start;
    uint32 end;
} address_range;
typedef struct name{
    char *language_tag; /* NULL terminated string identifying the language
                           used for this name. */
    char *name;         /* NULL terminated string containing the name. */
} name;
```

- `min_desired_addresses` and `max_desired_addresses` are 16 bit unsigned integers representing the number of addresses which should be allocated.
- `min_desired_start_time` and `max_desired_start_time` are 32 bit unsigned integers representing the desired start time of the allocation. It has the same format as the time given by the `time` function.
- `min_desired_lifetime` and `max_desired_lifetime` are 32 bit unsigned integers containing the number of seconds which the allocations should last, as counted from the start time.
- `server` is the IP address of the MADCAP server which the request for allocation should be sent to. If no server is provided, a server will be searched for in the scope where the allocation should take place.
- `multicast_address_set_lease` is a pointer to the struct where the lease should be placed if the allocation is successful. The format of the struct is

```
typedef struct lease{
    uint32 *addresses;      /* The set of addresses leased. */
    unsigned int num_addr; /* The number of addresses leased. */
    uint16 address_family; /* The address family of the lease. */
    uint32 lifetime;       /* How long are the addresses valid? */
    uint32 start_time;     /* When the lease start beeing valid. */
    uint8  *lease_id;      /* Unique identifier of this lease. */
    unsigned int id_len;   /* The length of the lease id. */
    uint32 server_address; /* The server which gave us the lease. */
} lease;
```

- `status` is a pointer to an integer where the status code will be returned. The code can be MC_SOK or MC_STRY_LATER. If the return value of the functions indicate success, this integer must still be checked to see if the allocation has succeeded, or should be attempted later.

All functions return 0 to indicate success, and −1 to indicate failure. The possible errors are MC_EFAMILY, MC_ESCOPE, MC_ENUMADDR, MC_ELTIME, MC_ESTIME, MC_ETIMECONF and MC_EFAILED. The error codes can be read in the manner described in section 3.4.

3.2 Changing the lifetime of an allocation

This function is called in order to change the lifetime of an allocation. The function is declared as follows.

```
int change_multicast_addr_lifetime(lease *multicast_address_set_lease,
                                   uint32 min_desired_lifetime,
                                   uint32 max_desired_lifetime,
                                   uint32 *lifetime);
```

The arguments are:

- `multicast_address_set_lease` is the identifier of the lease as received from the allocation function.
- `min_desired_lifetime` and `max_desired_lifetime` are 32 bit unsigned integers containing the number of seconds which the allocations should last, as counted from the start time.
- `lifetime` is a pointer to a 32 bit unsigned integer which will contain the new lifetime after the function has succeeded. The new lifetime will also be stored in the lease.

The function returns 0 to indicate success, and −1 to indicate failure. The possible errors are MC_EFAMILY, MC_ELTIME, MC_EFAILED and MC_ELEASE. The error codes can be read in the manner described in section 3.4.

3.3 Deallocating addresses

To release a lease this function should be called.

```
int deallocate_multicast_addr(lease *multicast_address_set_lease);
```

The only argument is the lease which should be released. The function returns 0 to indicate success, and −1 to indicate failure. The possible errors are MC_EFAMILY, MC_ELEASE and MC_EFAILED. The error codes can be read in the manner described in section 3.4.

3.4 Error handling

When any of the functions above return −1, that is an indication of an error. The global variable `mc_errno` is then set to indicate the type of error which has occurred. When a function has succeeded the value of `mc_errno` is undefined. There are two utility functions which can be used with the error handling. They are declared as below.

This is used to get a string describing the error passed as argument.

```
char *mc_strerror(int errnum);
```

This is used to print an error message consisting of the string passed as argument followed by a colon and the description of the current error and a newline.

```
void mc_perror(const char *s);
```

The descriptions of the error messages can be found in the file `madcap.c`.

4. Linking libmadcap in a program

The header file `madcap/madcap.h` should be included in the program, and the library should be passed to the linker. When using gcc this is done by giving the argument `-lmadcap`.