The Minet Socket Interface

The Minet socket interface provides a single interface for application programs to talk to the Minet network stack or to the kernel's network stack. The interface looks like a simplified version of Berkeley socket interface. When communicating with the kernel stack, it is merely a thin veneer on top of that interface. This permits you to write a program using Minet that you can test on top of the kernel stack to check that it works before trying to run it on top of the Minet stack. It also means that you can use the man pages for the non-Minet versions of the API functions to get more information. For example, to learn more about `minet_socket()`, you can check the man page for `socket`.

Compiling and Linking

The minet socket interface is included with the minet distribution. To create a new application simply

To compile, you must include the minet_socket header file in your source file as follows:

```
#include "minet_socket.h"
```

Furthermore, you must add the application to the Minet build process. This is done by adding the target object file to `minet/src/apps/Makefile.in`. Once this is done Minet will compile the application automatically and copy the executable to `minet/bin/`.

Return codes and errors

Each of the minet_ functions returns an integer. A negative return code denotes an error. You can retrieve the exact error, or print an informative error message using the following functions:

```
int minet_error();
int minet_perror(char *s);
```

Initializing and Deinitializing the Minet Socket Interface

Before you use the minet socket interface, you must initialize it. You can initialize it to run on top of the kernel stack or the Minet stack:

```
int minet_init(enum {MINET_KERNEL, MINET_USER} type);
```

When you are done using the Minet socket interface, you should deinitialize it:

```
int minet_deinit();
```

Creating A Socket

To create a socket, you can use a basic call that can only create PF_INET (internet) sockets:

```
int minet_socket(int type);
```

`type` must be either SOCK_STREAM (TCP) or SOCK_DGRAM (UDP).

Binding A Socket

You can bind a socket to an IP address (AF_INET) and port using the following call:

```
int minet_bind(int                 sockfd,
               struct sockaddr_in *myaddr);
```

Listening On A Socket

You can listen on a socket using the following call:

```
int minet_listen(int sockfd,
                 int backlog);
```

Accepting A Connection

You can accept a connection with the following call:

```
int minet_accept(int                 sockfd,
                 struct sockaddr_in *addr);
```

Connecting To A Remote Socket

To connect to a remote socket, you can use the following call:

```
int minet_connect(int                 sockfd,
                  struct sockaddr_in *addr);
```

Sending And Receiving

To send and receive messages, you can use read and write calls:

```
int minet_read(int fd,
               char *buf,
               int len);
int minet_write(int fd,
                char *buf,
                int len);
```

These calls return the number of bytes actually read or written. They will only work for sockets that are connected. If you want to send data on an unconnected socket (A UDP socket, for example), you should use the following calls:

```
int minet_sendto(int                 fd,
                 char               *buf,
                 int                 len,
                 struct sockaddr_in *to);

int minet_recvfrom(int                 fd,
                   char               *buf,
                   int                 len,
                   struct sockaddr_in *from)
```

Closing A Socket

To close a socket use the following call:

```
      int minet_close(int sockfd);
```

Select

The select call in the Minet socket interface is complicated by the fact that one might want to simultaneously select on both Minet sockets and on other, non-Minet file descriptors. For this reason, there are two Minet select calls.

The Minet select calls use `fd_set`s just like the Unix select call. Thus the following functions will work on Minet sockets:

```
      FD_CLR(int fd, fd_set *set);
      FD_ISSET(int fd, fd_set *set);
      FD_SET(int fd, fd_set *set);
      FD_ZERO(fd_set *set);
```

The basic Minet select call is used if you need to select on only Minet sockets:

```
      int minet_select(int             minet_maxfd,
                       fd_set         *minet_read_fds,
                       fd_set         *minet_write_fds,
                       fd_set         *minet_except_fds,
                       struct timeval *timeout);
```

If you want to select on both Minet sockets and non-Minet file descriptors, you can use the extended version of the call. Essentially, you pass in separate `fd_set`s for the non-Minet file descriptors:

```
      int minet_select_ex(int             minet_maxfd
                          fd_set         *minet_read_fds,
                          fd_set         *minet_write_fds,
                          fd_set         *minet_except_fds,
                          int             unix_maxfd,
                          fd_set         *unix_read_fds,
                          fd_set         *unix_write_fds,
                          fd_set         *unix_except_fds,
                          struct timeval *timeout);
```

Poll

Just like select, the Minet poll function comes in two flavors, one for polling Minet sockets:

```
int minet_poll(struct pollfd *minet_fds,
               int            num_minet_fds,
               int            timeout);
```

and one for polling both Minet sockets and non-Minet file descriptors:

```
int minet_poll_ex(struct pollfd *minet_fds,
                  int            num_minet_fds,
                  struct pollfd *unix_fds,
                  int            num_unix_fds,
                  int            timeout);
```

Utility Functions

You can set whether a Minet socket will be blocking or non-blocking using the following functions:

```
int minet_set_nonblocking(int sockfd);
int minet_set_blocking(int sockfd);
```

You can query whether a socket is ready for reading or writing using the following functions:

```
int minet_can_write_now(int sockfd);
int minet_can_read_now(int sockfd);
```