NAME

conet_init, conet_cleanup, conet_readsome, conet_read, conet_readln, conet_write, conet_printf, conet_new_conn, conet_close_conn, conet_set_timeo, conet_mod_conn, conet_socket, conet_connect, conet_accept, conet_create_conn, conet_events_wait, conet_events_dispatch

SYNOPSIS

#include <coronet.h>

int conet_init(void);
void conet_cleanup(void);
int conet_readsome(struct sk_conn * *conn***, void *** *buf* **, int** *n***);**
int conet_read(struct sk_conn * *conn***, void *** *buf* **, int** *n***);**
char *conet_readln(struct sk_conn * *conn***, int *** *lnsize***);**
int conet_write(struct sk_conn * *conn***, void const *** *buf* **, int** *n***);**
int conet_printf(struct sk_conn * *conn***, char const *** *fmt***, ...);**
struct sk_conn *conet_new_conn(int *sfd***, coroutine_t** *co***);**
void conet_close_conn(struct sk_conn * *conn***);**
int conet_set_timeo(struct sk_conn * *conn***, int** *timeo***);**
int conet_mod_conn(struct sk_conn * *conn***, unsigned int** *events***);**
int conet_socket(int *domain***, int** *type***, int** *protocol***);**
int conet_connect(struct sk_conn * *conn***, const struct sockaddr *** *serv_addr***, socklen_t** *addrlen***);**
int conet_accept(struct sk_conn * *conn***, struct sockaddr *** *addr***, int *** *addrlen***);**
struct sk_conn *conet_create_conn(int *domain***, int** *type***, int** *protocol***, coroutine_t** *co***);**
int conet_events_wait(int *timeo***);**
int conet_events_dispatch(int *evdmax***);**

DESCRIPTION

The **coronet** library implements an **epoll** and **coroutine** based library that allows for async operations over certain kinds of files (any file that supports **poll**(2) and the *O_NONBLOCK* **fcntl**(2) flag can be hosted - like sockets, pipes, ...). The **coronet** library uses the **epoll**(4) support available in the 2.6 series of **Linux** kernels (using a **glibc** version of 2.3.2 or newer), and the **libpcl** library for coroutine support (available as a package in many distributions). See the **AVAILABILITY** section and the **pcl**(3) man page for more information about coroutines and their management with **libpcl**.

FUNCTIONS

The following functions are defined:

int conet_init(void);

The **conet_init** function initializes the **coronet** library. It must be called before any other **coronet** function is called. It returns 0 in case of success, or a negative number in case of error.

void conet_cleanup(void);

The **conet_cleanup** function cleanups the **coronet** context. It should be called when the suer wants to free all the resources associated with the **coronet** library.

int conet_readsome(struct sk_conn * *conn***, void *** *buf* **, int** *n***);**

The **conet_readsome** function reads some data from the *conn* connection. Data will be stored in the *buf* buffer, and up to *n* bytes will be read. The function returns the number of bytes read (that can be lower than *n*), or a negative number in case of error. Zero can be also returned, to indicate

that we are at the end of file (or that the remote peer closed the connection, in case of a socket).

int conet_read(struct sk_conn ****conn*, void ****buf***, int** *n***);**

The **conet_read** function reads *n* bytes into the *buf* buffer, from the *conn* connection. The **conet_read** function tries to read *n* bytes, and does not return until *n* bytes are read, or an error occurred. The function returns the number of bytes read, or a number lower than *n* in case of error. A number lower then *n* can be also returned, to indicate that we are at the end of file (or that the remote peer closed the connection, in case of a socket).

int conet_write(struct sk_conn ****conn*, void const ****buf***, int** *n***);**

The **conet_write** function writes *n* bytes from the *buf* buffer, into the *conn* connection. The **conet_write** function tries to write *n* bytes, and does not return until *n* bytes are written, or an error occurred. The function returns the number of bytes written, or a number lower than *n* in case of error.

int conet_printf(struct sk_conn ****conn*, char const *** *fmt***, ...);**

The **conet_printf** function writes a formatted string to the *conn* connection. The function returns the number of bytes written, or a negative number in case of error.

struct sk_conn *conet_new_conn(int *sfd***, coroutine_t** *co***);**

The **conet_new_conn** function creates a new connection based on the *sfd* file descriptor and the *co* coroutine. The function returns a new connection pointer in case of success, or **NULL** in case of error.

void conet_close_conn(struct sk_conn ****conn*);**

The **conet_close_conn** function closes the connection passed into the *conn* parameter. A **close**(2) call will be performed of the file descriptor at this stage.

int conet_set_timeo(struct sk_conn ****conn*, int** *timeo***);**

The **conet_set_timeo** function sets the expiration timeout *timeo* for the async operations issued over the *conn* connection. The *timeo* timeout is in seconds. The expire timeout does not need to be seen as an high precision timing, since the expire operation is lazily done inside the **conet_events_dispatch** function. A one to two seconds resolution can be expected.

int conet_mod_conn(struct sk_conn ****conn*, unsigned int** *events***);**

The **conet_mod_conn** function modifies the interest event set for the connection *conn* to the *events* set. See **epoll_ctl**(2) for a detailed description of the supported event bits. The function returns 0 in case of success, or a negative number in case of error.

int conet_socket(int *domain***, int** *type***, int** *protocol***);**

The **conet_socket** function creates a new socket descriptor to be used inside the **coronet** library. The *domain*, *type* and *protocol* parameters are the same as the ones passed to the **socket**(2) function. The function returns the newly created socket descriptor, or -1 in case of error.

int conet_connect(struct sk_conn ****conn*, const struct sockaddr ****serv_addr***, socklen_t** *addrlen***);**

The **conet_connect** functions tries to establish a connection from *conn* to the remote address pointed by *serv_addr*, whose length is passed in the *addrlen* parameter. The function returns 0 in case of success, or a negative number in case of error.

int conet_accept(struct sk_conn ****conn*, struct sockaddr ****addr***, int ****addrlen***);**

The **conet_accept** function accepts a new connection into *conn* and stores the peer address in the *addr* parameter. The parameter *addrlen* stores the size of *addr* in input, and receives the size of *addr* in output. The function returns the newly accepted socket descriptor, or -1 in case of error.

struct sk_conn *conet_create_conn(int *domain***, int** *type***, int** *protocol***, coroutine_t** *co***);**

The **conet_create_conn** creates a new connection. This is basically a chain of a **conet_socket** and a **conet_new_conn** function calls. The function returns a new connection pointer in case of success, or **NULL** in case of error.

int conet_events_wait(int *timeo***);**

The **conet_events_wait** function waits for some events ready for all the async operations currently in flight on any of the file descriptors hosted by the **coronet** library. This is the only blocking call of all the **coronet** functions. The *timeo* specify a maximum time, in milliseconds, to wait for events. The function returns the current number of ready events currently available for dispatch, or a negative number in case of error.

int conet_events_dispatch(int *evdmax***);**

The **conet_events_dispatch** function dispatch up to *evdmax* events to the coroutines handling the async operations that became ready. If *evdmax* is lower or equal to zero, all the pending ready events will be dispatched. The function returns the number of dispatched events, or a negative number in case of error.

EXAMPLE

A few example usage of the **coronet** library are available inside the *test* subdirectory of the **coronet** package. An example of a trivial web server based on the **coronet** library is here reported inline:

```
#define _GNU_SOURCE
#include <sys/types.h>
#include <sys/time.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <fcntl.h>
#include <string.h>
#include <ctype.h>
#include <unistd.h>
#include <errno.h>
#include <stdarg.h>
#include <limits.h>
#include <signal.h>
```

```
#include <dirent.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/tcp.h>
#include <arpa/inet.h>
#include <arpa/nameser.h>
#include <netdb.h>
#include "coronet.h"


#define CNHD_EVWAIT_TIMEO 1000
#define CNHD_STKSIZE (1024 * 8)


static int stopsvr;
static char const *rootfs = ".";
static int svr_port = 80;
static int lsnbklog = 128;
static int stksize = CNHD_STKSIZE;
static unsigned long long conns, reqs, tbytes;


static int cnhd_set_cork(int fd, int v) {

        return setsockopt(fd, SOL_TCP, TCP_CORK, &v, sizeof(v));
}

static int cnhd_send_mem(struct sk_conn *conn, long size, char const *ver,
                         char const *cclose) {
        size_t csize, n;
        long msent;
        static char mbuf[1024 * 8];

        cnhd_set_cork(conn->sfd, 1);
        conet_printf(conn,
                     "%s 200 OK\r\n"
                     "Connection: %s\r\n"
                     "Content-Length: %ld\r\n"
                     "\r\n", ver, cclose, size);
        for (msent = 0; msent < size;) {
                csize = (size - msent) > sizeof(mbuf) ?
                        sizeof(mbuf): (size_t) (size - msent);
                if ((n = conet_write(conn, mbuf, csize)) > 0)
                        msent += n;
                if (n != csize)
                        break;
        }
        cnhd_set_cork(conn->sfd, 0);

        tbytes += msent;

        return msent == size ? 0: -1;
}
```

```
static int cnhd_send_doc(struct sk_conn *conn, char const *doc, char const *ver,
                         char const *cclose) {

        conet_printf(conn,
                     "%s 404 OK\r\n"
                     "Connection: %s\r\n"
                     "Content-Length: 0\r\n"
                     "\r\n", ver, cclose);

        return 0;
}

static int cnhd_send_url(struct sk_conn *conn, char const *doc, char const *ver,
                         char const *cclose) {
        int error;

        if (strncmp(doc, "/mem-", 5) == 0)
                error = cnhd_send_mem(conn, atol(doc + 5), ver, cclose);
        else
                error = cnhd_send_doc(conn, doc, ver, cclose);

        return error;
}

static void *cnhd_service(void *data) {
        int cfd = (int) (long) data;
        int cclose = 0, chunked, lsize, clen;
        char *req, *meth, *doc, *ver, *ln, *auxptr;
        struct sk_conn *conn;

        if ((conn = conet_new_conn(cfd, co_current())) == NULL)
                return NULL;
        while (!stopsvr && !cclose) {
                if ((req = conet_readln(conn, &lsize)) == NULL)
                        break;
                if ((meth = strtok_r(req, " ", &auxptr)) == NULL ||
                   (doc = strtok_r(NULL, " ", &auxptr)) == NULL ||
                   (ver = strtok_r(NULL, " \r\n", &auxptr)) == NULL ||
                   strcasecmp(meth, "GET") != 0) {
                        bad_request:
                        free(req);
                        conet_printf(conn,
                                     "HTTP/1.1 400 Bad request\r\n"
                                     "Connection: close\r\n"
                                     "Content-Length: 0\r\n"
                                     "\r\n");
                        break;
                }
                reqs++;
                cclose = strcasecmp(ver, "HTTP/1.1") != 0;
                for (clen = 0, chunked = 0;;) {
                        if ((ln = conet_readln(conn, &lsize)) == NULL)
                                break;
                        if (strcmp(ln, "\r\n") == 0) {
```

```
                                free(ln);
                                break;
                        }
                        if (strncasecmp(ln, "Content-Length:", 15) == 0) {
                                for (auxptr = ln + 15; *auxptr == ' '; auxptr++);
                                clen = atoi(auxptr);
                        } else if (strncasecmp(ln, "Connection:", 11) == 0) {
                                for (auxptr = ln + 11; *auxptr == ' '; auxptr++);
                                cclose = strncasecmp(auxptr, "close", 5) == 0;
                        } else if (strncasecmp(ln, "Transfer-Encoding:", 18) == 0) {
                                for (auxptr = ln + 18; *auxptr == ' '; auxptr++);
                                chunked = strncasecmp(auxptr, "chunked", 7) == 0;
                        }
                        free(ln);
                }
                /*
                 * Sorry, really stupid HTTP server here. Neither GET payload nor
                 * chunked encoding allowed.
                 */
                if (clen || chunked)
                        goto bad_request;
                cnhd_send_url(conn, doc, ver, cclose ? "close": "keep-alive");
                free(req);
        }
        conet_close_conn(conn);

        return data;
}

static void *cnhd_acceptor(void *data) {
        int sfd = (int) (long) data;
        int cfd, addrlen = sizeof(struct sockaddr_in);
        coroutine_t co;
        struct sk_conn *conn;
        struct sockaddr_in addr;

        if ((conn = conet_new_conn(sfd, co_current())) == NULL)
                return NULL;
        while (!stopsvr &&
             (cfd = conet_accept(conn, (struct sockaddr *) &addr,
                                 &addrlen)) != -1) {
                conns++;
                if ((co = co_create((void *) cnhd_service, (void *) (long) cfd, NULL,
                                    stksize)) == NULL) {
                        fprintf(stderr, "Unable to create coroutine\n");
                        close(cfd);
                } else
                        co_call(co);
        }
        conet_close_conn(conn);

        return data;
}
```

```
static void cnhd_sigint(int sig) {

	stopsvr++;
}

static void cnhd_usage(char const *prg) {

	fprintf(stderr, "Use: %s [-p PORT (%d)] [-r ROOTFS ('%s')] [-L LSNBKLOG (%d)]\n"
		"\t[-S STKSIZE (%d)] [-h]\n", prg, svr_port, lsnbklog, stksize);
}

int main(int ac, char **av) {
	int i, sfd, one = 1;
	coroutine_t co;
	struct linger ling = { 0, 0 };
	struct sockaddr_in addr;

	for (i = 1; i < ac; i++) {
		if (strcmp(av[i], "-r") == 0) {
			if (++i < ac)
				rootfs = av[i];
		} else if (strcmp(av[i], "-p") == 0) {
			if (++i < ac)
				svr_port = atoi(av[i]);
		} else if (strcmp(av[i], "-L") == 0) {
			if (++i < ac)
				lsnbklog = atol(av[i]);
		} else if (strcmp(av[i], "-S") == 0) {
			if (++i < ac)
				stksize = atoi(av[i]);
		} else {
			cnhd_usage(av[0]);
			return 1;
		}
	}
	signal(SIGINT, cnhd_sigint);
	signal(SIGPIPE, SIG_IGN);
	siginterrupt(SIGINT, 1);
	if (conet_init() < 0)
		return 1;
	if ((sfd = conet_socket(AF_INET, SOCK_STREAM, 0)) == -1) {
		conet_cleanup();
		return 2;
	}
	setsockopt(sfd, SOL_SOCKET, SO_REUSEADDR, &one, sizeof(one));
	setsockopt(sfd, SOL_SOCKET, SO_LINGER, &ling, sizeof(ling));

	addr.sin_family = AF_INET;
	addr.sin_port = htons(svr_port);
	addr.sin_addr.s_addr = htonl(INADDR_ANY);
	if (bind(sfd, (struct sockaddr *) &addr, sizeof(addr)) == -1) {
		perror("bind");
		close(sfd);
		conet_cleanup();
```

```
		return 3;
	}
	listen(sfd, lsnbklog);
	if ((co = co_create((void *) cnhd_acceptor, (void *) (long) sfd, NULL,
				stksize)) == NULL) {
		fprintf(stderr, "Unable to create coroutine\n");
		close(sfd);
		conet_cleanup();
		return 4;
	}
	co_call(co);

	while (!stopsvr) {
		conet_events_wait(CNHD_EVWAIT_TIMEO);
		conet_events_dispatch(0);
	}

	close(sfd);
	conet_cleanup();

	fprintf(stdout,
		"Connections .....: %llu\n"
		"Requests ........: %llu\n"
		"Total Bytes .....: %llu\n", conns, reqs, tbytes);

	return 0;
}
```

LICENSE

AUTHOR

Developed by Davide Libenzi <**davidel@xmailserver.org**>

AVAILABILITY

The latest version of **coronet** can be found at :

http://www.xmailserver.org/coronet-lib.html

The latest version of **libpcl** can be found at :

http://www.xmailserver.org/libpcl.html

BUGS

There are no known bugs. Bug reports and comments to Davide Libenzi <**davidel@xmailserver.org**>