Threaded networking

Let us revisit the socket programming examples, a client and a server for a simple echo protocol. The (slightly updated) `client` program takes a hostname and a port, connects to the host and port, and copies the standard input to the socket until end-of-file, and finally reads back the response and prints it.

client.c

```c
#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/socket.h>
#include <netdb.h>

#define BUF_SIZE 500

int main(int argc, char **argv)
{
        struct addrinfo hints, *result, *rp;
        int sfd, s;
        ssize_t len, nsent, ret;
        char buf[BUF_SIZE];

        if (argc != 3) {
                fprintf(stderr, "Usage: %s HOST PORT\n", argv[0]);
                exit(EXIT_FAILURE);
        }

        memset(&hints, 0, sizeof hints);
        hints.ai_socktype = SOCK_STREAM;

        s = getaddrinfo(argv[1], argv[2], &hints, &result);
        if (s != 0) {
                fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(s));
                exit(EXIT_FAILURE);
        }

        for (rp = result; rp; rp = rp->ai_next) {
                sfd = socket(rp->ai_family, rp->ai_socktype, rp->ai_protocol);
                if (sfd == -1)
                        continue;
                if (connect(sfd, rp->ai_addr, rp->ai_addrlen) == 0)
                        break;
                close(sfd);
        }

        freeaddrinfo(result);
        if (!rp) {
                fprintf(stderr, "Could not bind\n");
                exit(EXIT_FAILURE);
        }

        while (fgets(buf, sizeof buf, stdin)) {
                len = strlen(buf);
                for (nsent = 0; nsent < len; nsent += ret) {
                        ret = send(sfd, &buf[nsent], len - nsent, 0);
                        if (ret <= 0) {
                                if (ret < 0)
                                        perror("send");
                                break;
                        }
                }
        }

        shutdown(sfd, SHUT_WR);

        while ((ret = recv(sfd, buf, sizeof buf, 0)) > 0)
                fwrite(buf, ret, 1, stdout);
        if (ret < 0)
                perror("recv");

        close(sfd);
        return EXIT_SUCCESS;
}
```

The server accepts connections one at a time, and on each connection reads input and echoes it back as it is received.

`server.c`

```
#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/socket.h>
#include <netdb.h>

#define BUF_SIZE 500

void connection(int client)
{
        ssize_t nread, nsent, ret;
        char buf[BUF_SIZE];

        while ((nread = recv(client, buf, sizeof buf, 0)) > 0) {
                for (nsent = 0; nsent < nread; nsent += ret) {
                        ret = send(client, &buf[nsent], nread - nsent, 0);
                        if (ret <= 0) {
                                if (ret < 0)
                                        perror("send");
                                break;
                        }
                }
        }

        close(client);
}

int main(int argc, char **argv)
{
        struct addrinfo hints, *result, *rp;
        int sfd, client, s;

        if (argc != 2) {
                fprintf(stderr, "Usage: %s PORT\n", argv[0]);
                exit(EXIT_FAILURE);
        }

        memset(&hints, 0, sizeof hints);
        hints.ai_socktype = SOCK_STREAM;
        hints.ai_flags = AI_PASSIVE;

        s = getaddrinfo(NULL, argv[1], &hints, &result);
        if (s != 0) {
                fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(s));
                exit(EXIT_FAILURE);
        }

        for (rp = result; rp; rp = rp->ai_next) {
                sfd = socket(rp->ai_family, rp->ai_socktype, rp->ai_protocol);
                if (sfd == -1)
                        continue;
                if (bind(sfd, rp->ai_addr, rp->ai_addrlen) == 0)
                        break;
                close(sfd);
        }

        freeaddrinfo(result);
        if (!rp) {
                fprintf(stderr, "Could not bind\n");
                exit(EXIT_FAILURE);
        }

        if (listen(sfd, 10) == -1) {
                perror("listen");
                close(sfd);
                exit(EXIT_FAILURE);
        }
        while ((client = accept(sfd, NULL, NULL)) != -1)
                connection(client);
        perror("accept");

        close(sfd);
        exit(EXIT_FAILURE);
}
```

Let us first update the client so that the responses from the server appear as soon as possible, rather than first sending everything and then reading everything in turn. This will require threading, with one thread for the sending and one thread for the receiving.

client_threaded.c

```
#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/socket.h>
#include <netdb.h>
#include <pthread.h>

#define BUF_SIZE 500

int sfd;

void *connection(void *arg)
{
        ssize_t len, nsent, ret;
        char buf[BUF_SIZE];

        while (fgets(buf, sizeof buf, stdin)) {
                len = strlen(buf);
                for (nsent = 0; nsent < len; nsent += ret) {
                        ret = send(sfd, &buf[nsent], len - nsent, 0);
                        if (ret <= 0) {
                                if (ret < 0)
                                        perror("send");
                                break;
                        }
                }
        }

        shutdown(sfd, SHUT_WR);
        return arg;
}

int main(int argc, char **argv)
```

```c
{
        struct addrinfo hints, *result, *rp;
        int s;
        ssize_t ret;
        char buf[BUF_SIZE];
        pthread_t tid;

        if (argc != 3) {
                fprintf(stderr, "Usage: %s HOST PORT\n", argv[0]);
                exit(EXIT_FAILURE);
        }

        memset(&hints, 0, sizeof hints);
        hints.ai_socktype = SOCK_STREAM;

        s = getaddrinfo(argv[1], argv[2], &hints, &result);
        if (s != 0) {
                fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(s));
                exit(EXIT_FAILURE);
        }

        for (rp = result; rp; rp = rp->ai_next) {
                sfd = socket(rp->ai_family, rp->ai_socktype, rp->ai_protocol);
                if (sfd == -1)
                        continue;
                if (connect(sfd, rp->ai_addr, rp->ai_addrlen) == 0)
                        break;
                close(sfd);
        }

        freeaddrinfo(result);
        if (!rp) {
                fprintf(stderr, "Could not bind\n");
                exit(EXIT_FAILURE);
        }

        pthread_create(&tid, NULL, connection, NULL);

        while ((ret = recv(sfd, buf, sizeof buf, 0)) > 0)
                fwrite(buf, ret, 1, stdout);
        if (ret < 0)
                perror("recv");

        pthread_join(tid, NULL);
        close(sfd);
        return EXIT_SUCCESS;
}
```

Note what happens, however, if multiple clients try to connect simultaneously. Because the server has only one thread, it can only wait for one thing at a time, so only one connection can be accepted at a time. The second client will not successfully connect until the first is completely closed. The server will need a thread per client.

server_threaded.c

```c
#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/socket.h>
#include <netdb.h>
#include <pthread.h>

#define BUF_SIZE 500

void *connection(int client)
{
	ssize_t nread, nsent, ret;
	char buf[BUF_SIZE];

	while ((nread = recv(client, buf, sizeof buf, 0)) > 0) {
		for (nsent = 0; nsent < nread; nsent += ret) {
			ret = send(client, &buf[nsent], nread - nsent, 0);
			if (ret <= 0) {
				if (ret < 0)
					perror("send");
				break;
			}
		}
	}

	close(client);
	return NULL;
}

int main(int argc, char **argv)
{
	struct addrinfo hints, *result, *rp;
	int sfd, client, s;
	pthread_attr_t attr;
	pthread_t tid;

	if (argc != 2) {
		fprintf(stderr, "Usage: %s PORT\n", argv[0]);
		exit(EXIT_FAILURE);
	}
```

```
        pthread_attr_init(&attr);
        pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);

        memset(&hints, 0, sizeof hints);
        hints.ai_socktype = SOCK_STREAM;
        hints.ai_flags = AI_PASSIVE;

        s = getaddrinfo(NULL, argv[1], &hints, &result);
        if (s != 0) {
                fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(s));
                exit(EXIT_FAILURE);
        }

        for (rp = result; rp; rp = rp->ai_next) {
                sfd = socket(rp->ai_family, rp->ai_socktype, rp->ai_protocol);
                if (sfd == -1)
                        continue;
                if (bind(sfd, rp->ai_addr, rp->ai_addrlen) == 0)
                        break;
                close(sfd);
        }

        freeaddrinfo(result);
        if (!rp) {
                fprintf(stderr, "Could not bind\n");
                exit(EXIT_FAILURE);
        }

        if (listen(sfd, 10) == -1) {
                perror("listen");
                close(sfd);
                exit(EXIT_FAILURE);
        }

        while ((client = accept(sfd, NULL, NULL)) != -1)
                pthread_create(&tid, &attr, (void *(*)(void *))connection,
                               (void *)client);
        perror("accept");

        pthread_attr_destroy(&attr);
        close(sfd);
        exit(EXIT_FAILURE);
}
```