

Project 4C

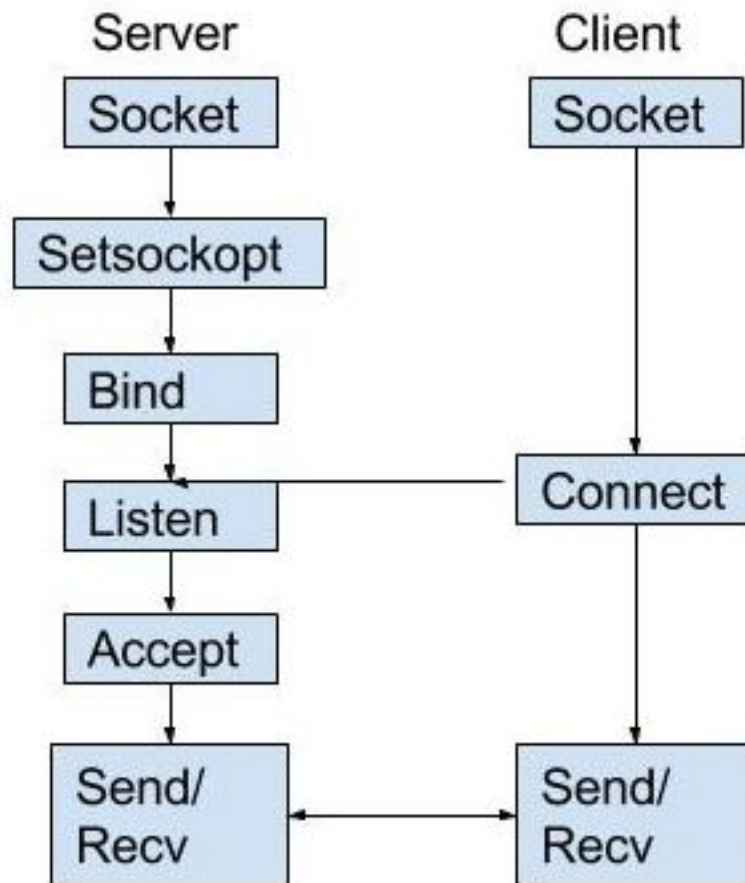
Alexandre Tiard

CS111

Sending the log over a secure channel

- Remove the button from 4B
- Commands will be the same
- Your program will accept additional parameters, that you will need for the connexion
- You will send the logfile produced by your Beaglebone to a server (through a TCP connection)
 - Optionally, you will use Openssl to perform encryption on that channel

Socket programming : tutorial



The good news is that we handled the server for you: you only have to create the client

socket(2)

Goal: Create an endpoint for communication

Success: Socket descriptor // **Error:** -1

int socket(int *domain*, int *type*, int *protocol*)

Communication Domain:
AF_INET : IPv4
AF_INET6 : IPv6
AF_LOCAL : local

Communication type:
SOCK_STREAM : TCP
SOCK_DGRAM : UDP

IP protocol value:
- If set to 0, OS
chooses protocol

This returns a file descriptor, similar to a pipe. However, sockets can handle bidirectional communication, so you only need 1 fd, not 2!

connect(2)

Goal: Connect to a remote host

Success: Socket descriptor // **Error:** -1

Socket for outbound connection (returned by socket())

```
int connect(int sockfd, const struct sockaddr *addr,
            socklen_t addrlen);
```

Length of the address structure (in bytes):
- sizeof(struct sockaddr)

Structure containing server's IP
address and port number

```
struct sockaddr_in {
    sa_family_t    sin_family; /* address family: AF_INET */
    in_port_t      sin_port;   /* port in network byte order */
    struct in_addr sin_addr;   /* internet address */
};
```

Use:

- atoi()
- htons()

```
/* Internet address. */
struct in_addr {
    uint32_t s_addr; /* address in network byte order */
};
```

Initialize to 0:
- bzero()

connect(2)

Why is the port number necessary?

connect(2)

Why is the port number necessary?

- To identify the specific process we want to interact with
- You have to use a port number above 1024, below are reserved. Some examples:
 - 20/21 FTP
 - 80 HTTP
 - 156 SQL Server
 - 443 HTTPS
 - ...
- In this project, the port number will be passed as an argument

gethostbyname(3)

Goal : Identify target structure / host port

struct hostent *gethostbyname(const char **name*)

```
struct hostent {  
    char    *h_name;           /* official name of host */  
    char    **h_aliases;      /* alias list */  
    int      h_addrtype;      /* host address type */  
    int      h_length;        /* length of address */  
    char    **h_addr_list;    /* list of addresses */  
}
```

Example: Local
struct hostent *h;
h = gethostbyname("localhost");
printf("Host name : %s \n", (*h).h_name)

This system call allows you to determine who you will communicate with. Pass this information to the connect() call to finish setting up your socket.

Secure Socket Layer(SSL) Library

- Difference between 'http' and 'https' is the use of ssl
- SSL provides:
 - Data encryption
 - Data authentication
 - Data Integrity
- This requires the use of keys
 - Numbers (typically 128/256 bits)
 - Combined with the message (using algorithms such as RSA)
- TLS : Based on SSL, was developed as replacement
 - When people say SSL they generally mean TLS
- In this project you will communicate with a server over a secure connection

Using the library

Before you can use any of the functions in the library, you need to:

- Install it (`sudo apt-get install libssl-dev`)
 - Your beaglebone might already have it installed
- Add the headers:
 - `openssl/ssl.h` and `openssl/err.h`
- Initialize the library:
 - `SSL_library_init()`
- Add algorithms to an internal table:
 - `OPENSSL_add_all_algorithms()`
- Load error strings:
 - `SSL_load_error_strings()`
- Compile with `-lssl -lcrypto` flags

Creating a context

- Encrypted connection requires a context object to be created (SSL_CTX)
- It is a framework allowing SSL/TLS
- You have to specify which connection method to use
 - Which version of SSL/TLS
 - Whether you want a generic connection, or server/client only
 - You can use TLSv1_client_method()
- `SSL_CTX *SSL_CTX_new(const SSL_METHOD method)`
- Returns a pointer to an SSL_CTX object upon success, NULL otherwise

SSL_new

Now that we have the framework set up -> create the structure that will hold connection data:

- `SSL *SSL_new(SSL_CTX *ctx)`
 - Creates an SSL structure, holding the data of connection
 - Inherits the settings of the underlying context *ctx
- Returns a pointer to the structure on success, NULL otherwise

SSL_set_fd

- What are we communicating with? Where does our encrypted output go ?
 - Through a socket onto the host
- A socket is handled by a file descriptor
 - Specify the socket to use with this function
- `Int SSL_set_fd(SSL *ssl, int fd)`
 - Argument 1 : SSL structure previously created
 - Argument 2 : socket file descriptor
 - Returns 1 on success, 0 on error
- Note: BIO is an I/O abstraction
 - used to create an (underlying) interface between ssl and fd
 - is automatically generated by this call.
 - Inherits blocking/nonblocking behavior from fd

SSL_connect

- We now know where our output is going locally (to the socket) and we linked the SSL object to encrypt it.
 - We now need to connect to transmit data
 - This is done by initiating a TLS/SSL handshake with the server
- `int SSL_connect(SSL *ssl)`
 - Takes as an argument the previously set SSL structure
 - Returns 1 on success, 0 on failure (controlled shutdown)
 - Returns <0 on fatal protocol-level error
- Rmk: the behavior of this function depends on the underlying BIO (such as blocking/non-blocking)

read/write

We now want to read commands/send output:

- `int SSL_write(SSL *ssl, const void *buf, int num)`
- `int SSL_read(SSL *ssl, const void *buf, int num)`
 - Write/Read num bytes from the specified ssl to/from the buffer buf
 - Returns >0 on success, <=0 on error
- Rmk: First thing that you have to send is your ID!

shutdown

- Before exiting, shutdown the client and free the SSL structure:
- `SSL_shutdown(SSL *ssl)`
- `SSL_free(SSL *ssl)`

Don't forget this step, we will check that you close the connection properly!