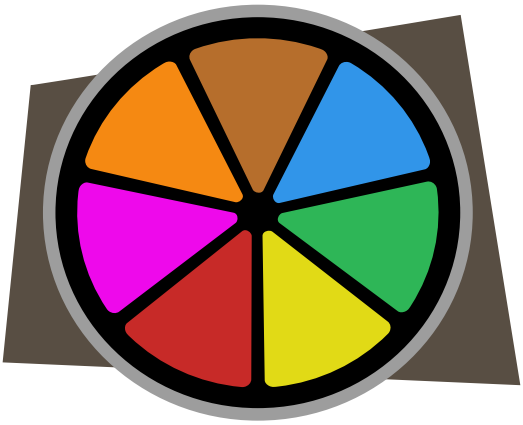
Preliminary Questions

Xarxes, Grau d'Enginyeria Informàtica

Document version: 1.0
Date: Feb. 2020



Universitat Autònoma de Barcelona

Departament d'Enginyeria de la Informació i de les Comunicacions

Document Information

Document identifier: preliminary_questions.md
Title: Preliminary Questions
Number of pages: 5
Date: Feb. 2020
Author(s): Ian Blanes, Rubén Martínez-Vidal, Angel Elbaz, Kevin Chow.

Change Control

Revision	Date	Changes
1.0	Feb. 2020	(initial version)

Contents

1 Introduction

This document contains various questions related to the **POSIX socket API**. Students should carefully study and answer these questions. After successfully completing this assignment, students should be knowledgeable of most of the quirks and details of the API.

2 How to answer the questions

These questions must be answered having application *portability* in mind. This means that the correct answer is the one that does not rely on the details of any specific implementation.

Most answers SHOULD only contain information from authoritative sources. The authoritative source will most likely be the POSIX.1-2017 specification available at `https://pubs.opengroup.org/onlinepubs/9699919799/`.

Tutorials and blogs often describe specific implementations and often contain small mistakes. They may be useful to help figure out how to answer some questions. However, these sources are not authoritative, and the equivalent information needs to be sought in authoritatve sources.

However, there will be cases where authoritative sources are not available. Such as questions 4 and 5.

A few examples are available at the end of this document.

3 Deliverable

The deliverable for this assignment is **a single `.odt` file** to be delivered in the Campus Virtual.

NOTES:

- A template is provided.
- It is not necessary that all members of a group contribute to the writing of each answer.
- **Each answer must be verified** to ensure its correctness by some group member, preferably, someone different than the one who wrote it.
- **Each answer must include information of the person who wrote it and the (other) person who verified it** (see template).

4 Questions

4.1 Data types and language issues

1. What is the function `htons` used for?
2. What is `uint32_t`?
3. What is the difference between `ssize_t` and `size_t`?
4. What is the value of `b` in the following piece of code?

```
const ssize_t a = -1;
const size_t b = (size_t) a;
```

5. What is the value of `b` in the following piece of code?

```
const uint32_t a = 256;
const uint8_t b = (uint8_t) a;
```

4.2 Sockets

6. What is a *file descriptor*?
7. What values for the `protocol` parameter of the `socket` function are available for `AF_INET` in POSIX? What does it mean that the default value for `protocol` in `socket` is *implementation defined*?
8. Give a piece of code that creates a TCP socket.

9. Give a piece of code that creates a UDP socket.
10. How do we close a socket?
11. What happens if we leave it open?
12. If we never close any sockets, we may eventually receive an `EMFILE` error code in `socket`. Why?

4.3 Error handling

13. What is `errno`? How is it used?
14. Why is the `perror` function used? Show a small example.
15. Can we modify `errno` ourselves in our code? Should we?
16. Which errors can result from a `socket` function call? And from `sendto`? (do not describe them the error codes, just mention them: `EAGAIN`)
17. What happens if a program terminates (e.g., `exit(EXIT_FAILURE)`) while still having some open sockets (not having called `close`)?

4.4 Binding

18. What does `bind` do?
19. What is the difference between `sockaddr`, `sockaddr_in`, `sockaddr_in6`, and {`sockaddr_storage`}?
20. Why is there an `addrlen` parameter in `bind`?
21. What is `in_port_t`?
22. Which of the following pieces of code regarding port `8080` is the right one: `in_port_t port = htons(8080)`, `in_port_t port = ntohs(8080)`, or `in_port_t port = 8080`? Why?
23. Give a proper initialization code for a `sockaddr_in` structure for port `8080` and address `127.0.0.0.1`.
24. Give a proper initialization code for a `sockaddr_in` structure for port `8080` and *any* address.

4.5 Memory problems

25. What is a null pointer?
26. What does it mean when a program receives a SIGSEGV?
27. What is *valgrind* and how can we use it to debug memory problems?
28. Why is it a good practice to `free` all allocated memory before calling `exit(EXIT_SUCCESS)`? (hint: valgrind)

4.6 Listening and connecting

29. What does `connect` do?
30. Why does `connect` take one `addr` parameter?
31. Why do we need to call `accept` after `listen`?
32. Why does `accept` take one `addr` parameter?
33. In the following piece of code, are `a` and `b` equal? Why?

```
int b = accept(a, ...);
```

34. What happens if we `listen` or `connect` to an unbound socket? (`bind` has not been called).

4.7 Fork server

35. On a forking server, after `accept` and `fork`, why should the parent close the socket returned by `accept`? (hint: resource exhaustion)

4.8 Non-blocking input and output

36. What is non-blocking I/O?
37. How can we set a socket to perform non-blocking I/O? (give code)
38. What happens with a `read` on a blocking socket if all the requested data is not available?
39. What happens with a `write` on a blocking socket if the socket buffer is full?
40. What happens with a `read` on a non-blocking socket if all the requested data is not available? (hint: there are two possibilities)
41. What happens with a `write` on a non-blocking socket if the data to write does not fit in the socket buffer? (hint: there are two possibilities)

4.9 Polling

42. How is the function `poll` used?
43. Can we mix file descriptors for sockets and disk files in a single `poll` call?
44. Which events can report a `poll` call for `fds[i].events = POLLIN;`?

4.10 Advanced

45. Are socket options inherited from a listening socket with `accept`?
46. Which size is the option for `SO_RCVLOWAT`?
47. After `poll` signals `revents = POLLOUT` on a non-blocking socket, will a very large send block execution?
48. Can `getaddrinfo` return an empty linked list?

5 Examples

5.1 Example 1

Question: Which header files are necessary for the `socket` function?

5.1.1 Incorrect answer

The following header files are required:

```c
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <arpa/inet.h>
//#include <fcntl.h>
#include <unistd.h>
```

Source: `https://stackoverflow.com/questions/9198608/simple-socket-programming-code-working`

5.1.2 Incorrect answer

Headers `sys/types.h` and `sys/socket.h` are needed.

Source: `http://man7.org/linux/man-pages/man2/socket.2.html`

5.1.3 Correct answer

Only the inclusion of the `sys/socket.h` headers is necessary. However, the `sys/types.h` header was required for some historical implementations.

Sources: `https://pubs.opengroup.org/onlinepubs/9699919799/basedefs/sys_socket.h.html` and `http://man7.org/linux/man-pages/man2/socket.2.html`

5.2 Example 2

Question: Which value shall we use for the `domain` field while creating an IPv4 socket? `PF_INET` or `AF_INET`?

5.2.1 Incorrect answer

We should use `PF_INET` in the `socket()` call because it refers to a protocol, and `AF_INET` in the `addrsock_in` structure because it refers to an address.

Source: `https://stackoverflow.com/questions/6729366/what-is-the-difference-between-af-inet-and-pf-inet-in-socket-programming`

5.2.2 Correct answer

The constant `PF_INET` is not defined in POSIX, and we should not use it. The constant `AF_INET` is defined as

> Internet domain sockets for use with IPv4 addresses.

We must use `AF_INET`.

Sources: `https://pubs.opengroup.org/onlinepubs/009695399/basedefs/sys/socket.h.html`

5.3 Example 3

Question: Why is the following piece of code incorrect?

```
//...
static const uint8_t MSG_OK = 1;
uint8_t buffer[8];
recv(sockfd1, buffer, 8, 0);

if (buffer[0] == MSG_OK) {
    // Do something
} else {
    // Do something else
}
```

5.3.1 Correct answer

The function `recv` returns an `ssize_t` value that must be checked for errors before assuming that the function succeeded and that `buffer[0]` contains valid data.

Source: `https://pubs.opengroup.org/onlinepubs/9699919799.2016edition/functions/recv.html`