(note: change tracking is enabled)

Q1 What is the function *htons* used for? Writing: GPB Verification: JMP

The **htons** function converts an IP port number (16-bits and 32-bits quantities) in host byte order to the IP port number, with the same length, in network byte order.

Sources: https://docs.microsoft.com/en-us/windows/win32/api/winsock/nf-winsock-htons, https://pubs.opengroup.org/onlinepubs/9699919799/, https://www.ibm.com/support/knowledge-center/en/SSB27U_6.4.0/com.ibm.zvm.v640.kiml0/asonetw.htm

Q2 What is *uint32_t*? Writing: JMP Verification: GPB

It's a numeric type which is an unsigned integer of 32 bits. Its range of numbers goes from 0 to 2^{32}-1.

Sources: https://stackoverflow.com/questions/48833976/what-is-uint32-t

Q3 What is the differences between *ssize_t* and *size_t*? Writing: JMP Verification: GPB

ssize_t: It is a signed integer type
size_t: It is an unsigned integer type.

Sources: https://pubs.opengroup.org/onlinepubs/9699919799/

Q4 What is the value of b in following piece of code? Writing: JMP Verification: GPB

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

As in *size_t* there aren't negatives values. Value "-1" in binary is all 1s when your change the type of variable, it get the most value possible. If the size is 32 bits, the maximun value is 2^32 -1.

Sources:

Q5 What is the value of b in following piece of code? Writing: JMP Verification: GPB

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

uint8_t can have to 2^8 - 1 = 255. The number 256 (*uint32_t*) in binary is 1 and followed by 8 0s, thus when you change for *uint8_t*, how it only can have 8 digits, the value is 0.

Sources:

Q6 What is a *file descriptor*? Writing: JMP Verification: GPB

It's a structure of data resident in the core used to access a file or other input/output resource. It is a non-negative integer.

Sources: https://en.wikipedia.org/wiki/File_descriptor, https://es.wikipedia.org/wiki/De-scriptor_de_archivo

Q7 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*? Writing: GPB Verification: JMP

The values you can use are 0 or IPPROTO_TCP for TCP sockets and 0 or IPPROTO_ UDP for UDP sockets.

Sources: http://man7.org/linux/man-pages/man7/ip.7.html

Q8 Give a piece of code that creates a TCP socket Writing: GPB Verification: JMP

```
  tcp_socket = socket(AF_INET, SOCK_STREAM, 0);
```

Sources: http://man7.org/linux/man-pages/man7/ip.7.html

Q9 Give a piece of code that creates a UDP socket Writing: GPB Verification: JMP

```
udp_socket = socket(AF_INET, SOCK_DGRAM, 0);
```

Sources: http://man7.org/linux/man-pages/man7/ip.7.html

Q10 How do we close a socket? Writing: JMP Verification: GPB

We close a socket using the following command: close(socket);

Sources: https://stackoverflow.com/questions/8051863/how-can-i-close-the-socket-in-a-proper-way

Q11 What happens if we leave it open? Writing: GPB Verification: JMP

It can be dangerous because a socket is an access to our computer, and maybe somebody can get information from your computer.

Sources:

Q12 If we never close any sockets, we may eventually receive an EMFILE error code in socket. Why? Writing: GPB Verification: JMP

EMFILE error code appears when we got too much sockets opened. If we want it to disappear, we must close some sockets.

Sources: https://stackoverflow.com/questions/8965606/node-and-error-emfile-too-many-open-files

Q13 What is *errno*? How is it used? Writing: GPB Verification: JMP

errno is an integer variable that contains the number of the last error.
It is useful because we can know if any error occurred checking what number it contains.

Sources: http://man7.org/linux/man-pages/man3/errno.3.html

Q14 Why is the *perror function* used? Show a small example. Writing: GPB Verification: JMP

It is used to print the number contained in errno variable described in last question. It also translate that number to be understandable by humans.
Example: `perror("Error: ");`

Sources: http://man7.org/linux/man-pages/man3/perror.3.html
https://www.tutorialspoint.com/c_standard_library/c_function_perror.htm

Q15 Can we modify *errno ourselves* in our code? Should we? Writing: GPB Verification: JMP

Yes we can. It is useful because we can change the value to 0 and the check if it changes, meaning that there is an error.

Sources: http://man7.org/linux/man-pages/man3/errno.3.html

Q16 Which errors can result from a *socket* function call? And from *sendto*? (do not describe them the error codes, just mention them: EAGAIN) Writing: JMP Verification: GPB

socket: EACCES, EAFNOSUPPORT,EINVAL,EMFILE, ENOBUFS o ENOMEM, EPROTONOSUPPORT
sendto: EACCES, EAGAIN o EWOULDBLOCK, EBADF, ECONNRESET, EDESTADDRREQ, EFAULT, EINTR, EINVAL, EISCONN, EMSGSIZE, ENOBUFS, ENOMEM, ENOTCONN, ENOTSOCK, EOPNOTSUPP, EPIPE

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

Q17 What happens if a program terminates (e. g., exit (EXIT_FAILURE)) while still having some open sockets (not having called close)? Writing: JMP Verification: GPB

If the the program terminates, the sockets are closed too, but they could be damaged.

Sources:

Q18 What does *bind* do? Writing: GPB Verification: JMP

bind() assigns the address specified to the socket referred to by the file descriptor.

Sources: http://man7.org/linux/man-pages/man2/bind.2.html

Q19	What is the difference between *sockaddr, sockaddr_in, sock-addr_in6,* and {*sockaddr_storage*}?	Writing: JMP	Verification: GPB

The *sockaddr* structure varies depending on the protocol selected. The *sockaddr_in* and *sock-addr_in6* are the same, they are a structure specifies a transport address and port for the AF_INET and AF_INET6 address families, respectively. The last, *sockaddr_storage*, is a structure sotore socket address information.

Sources: https://docs.microsoft.com/en-us/windows/win32/winsock/sockaddr-2, https://docs.microsoft.com/en-us/windows/win32/api/ws2def/ns-ws2def-sockaddr_in, https://riot-os.org/api/structsockaddr__in6.html, https://docs.microsoft.com/en-us/windows/win32/api/ws2ipdef/ns-ws2ipdef-sockaddr_in6_lh, https://docs.microsoft.com/en-us/previous-versions/windows/desktop/legacy/ms740504(v%3Dvs.85)

Q20	Why is there an *addrlen* parameter in bind?	Writing: JMP	Verification: GPB

It specifies the size (bytes) of the address structure pointed to by *addr.*

Sources: http://man7.org/linux/man-pages/man2/bind.2.html

Q21	What is *in_port_t*?	Writing: JMP	Verification: GPB

It is an unsignal integral type of exactly 16 bits. It is equivalent to *uint16_t*. Its range of number goes from 0 to 2^16 -1.

Sources: https://pubs.opengroup.org/onlinepubs/007908799/xns/netinetin.h.html,

Q22	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?	Writing: GPB	Verification: JMP

in_port_t port = htons(8080), because we need to transform a host byte order to network byte order.

Sources: https://linux.die.net/man/3/htons

Q23	Give a proper initialization code for a sockaddr_in structure for port 8080 and address 127.0.0.0.1.	Writing: GPB	Verification: JMP

```
#include <netinet/in.h>
#include <arpa/inet.h>

struct sockaddr_in s{
    sa_family_t    AF_INET;
    in_port_t      htons(8080);
    struct in_addr  inet_addr("127.0.0.0.1") ;
}
```

Sources: http://man7.org/linux/man-pages/man7/ip.7.html
http://man7.org/linux/man-pages/man3/inet_addr.3.html

Q24 Give a proper initialization code for a sockaddr_in structure for port 8080 and any address. Writing: GPB Verification: JMP

```
#include <netinet/in.h>
#include <arpa/inet.h>

struct sockaddr_in s{
    sa_family_t    AF_INET;
    in_port_t      htons(8080);
    struct in_addr  INADDR_ANY;
}
```

Sources: http://man7.org/linux/man-pages/man7/ip.7.html
http://man7.org/linux/man-pages/man3/inet_addr.3.html

Q25 What is a null pointer? Writing: JMP Verification: GPB

has a value saved for indicating that the pointer or reference does not refer to a valid object.

Sources: https://en.wikipedia.org/wiki/Null_pointer

Q26 What does it mean when a program receives a SIGSEGV? Writing: GPB Verification: JMP

It means that the program has attempted to access a restricted area of memory.

Sources: https://en.wikipedia.org/wiki/Segmentation_fault

Q27 What is *valgrind* and how can we use it to debug memory problems? Writing: JMP Verification: GPB

It is a tool for debugging and profiling executables.
For debug memory problems you can use the following command:

```
valgrind --leak-check=yes myprog arg1 arg2
```

It will show you a detailed memory detector.

Sources: https://valgrind.org/docs/manual/quick-start.html

Q28 Why is it a good practice to free all allocated memory before calling *exit(EXIT_SUCCESS)*? (hint: valgrind) Writing: JMP Verification: GPB

Yes, when allocated memory is freed up, powerful tools like valgrind can be used to check for memory leaks in the rest of code no false positives which the *malloc()* show.

Sources:

Q29 Writing: JMP Verification: GPB

What does connect do?

Initiate a connection on a socket.

Sources: http://man7.org/linux/man-pages/man2/connect.2.html

Q30 Writing: JMP Verification: GPB

Why does *connect* take one *addr* parameter?

Because it connects the socket specified by the file descriptor to the address specified by *addr*.

Sources: http://man7.org/linux/man-pages/man2/connect.2.html

Q31 Writing: GPB Verification: JMP

Why do we need to call *accept* after *listen*?

We need use because we want to accept the incoming connection request to the specified socket (listening socket).

Sources: http://man7.org/linux/man-pages/man2/accept.2.html

Q32 Writing: GPB Verification: JMP

Why does *accept* take one *addr* parameter?

Because it contains the address of the listening socket.

Sources: http://man7.org/linux/man-pages/man2/accept.2.html

Q33 Writing: GPB Verification: JMP

In the following piece of code, are a and b equal?
Why?

int b= accept(a, …);

They are not, because 'a' contains the listening socket fd, on the other hand 'b' contains the return of the function, which is a new fd for a new socket.

Sources: Sources: http://man7.org/linux/man-pages/man2/accept.2.html

Q34 Writing: JMP Verification: GPB

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

You can not work with this socket because bind has not been called and it does not operational.

Sources: Sources: http://man7.org/linux/man-pages/man2/bind.2.html

Q35 On a forking server, after *accept* and *fork*, why should the parent close the socket returned by *accept*? (hint: resource exhaustion) Writing: JMP Verification: GPB

The accept() function creates a new server that has the same characteristics than the father. If we do it several times it will cause resource exhaustion (because we create unnecessary sockets).

Sources: https://pubs.opengroup.org/onlinepubs/009695399/functions/accept.html, http://man7.org/linux/man-pages/man2/fork.2.html,

Q36 What is non-blocking I/O? Writing: GPB Verification: JMP

It is a kind of I/O which offers services that don't make the programs that use them wait.

Sources: Operative Systems subject

Q37 How can we set a socket to perform non-blocking I/O? (give code) Writing: GPB Verification: JMP

In order to perform non-blocking I/O we must use select() function. example code:

```
fd_set d;
int socketServer;
int socketClient[10];
int nClients;
...
FD_ZERO (&d);
FD_SET (socketServer, &d);
for (i=0; i<nClients; i++)
        FD_SET (socketClient[i], &d);
...
select (max+1, &d, NULL, NULL, NULL);
```

Sources: http://man7.org/linux/man-pages/man2/select.2.html http://www.chuidiang.org/clinux/sockets/socketselect.php

Q38 What happens with a *read* on a blocking socket if all the requested data is not available? Writing: GPB Verification: JMP

While the requested data is not available, the socke will remain waiting.

Sources: https://www.ibm.com/support/knowledgecenter/en/SSLTBW_2.2.0/com.ibm.zos.v2r2.hala001/orgblockasyn.htm

Q39 What happens with a *write* on a blocking socket if the socket buffer is full? Writing: GPB Verification: JMP

It will wait until the buffer has space.

Sources: https://www.ibm.com/support/knowledgecenter/en/SSLTBW_2.2.0/com.ibm.zos.v2r2.hala001/orgblockasyn.htm

Q40 What happens with a *read* on a non-blocking socket if all the requested data is not available? (hint: there are two possibilities) Writing: GPB Verification: JMP

It will return EWOULDBLOCK error number (select() exception: READ)

Sources: https://www.ibm.com/support/knowledgecenter/en/SSLTBW_2.2.0/com.ibm.zos.v2r2.hala001/orgblockasyn.htm

Q41 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) Writing: GPB Verification: JMP

It will return EWOULDBLOCK error number (select() exception: WRITE)

Sources: https://www.ibm.com/support/knowledgecenter/en/SSLTBW_2.2.0/com.ibm.zos.v2r2.hala001/orgblockasyn.htm

Q42 How is the function *poll* used? Writing: JMP Verification: GPB

The function *poll()* allow a program to use a file descriptor and waiting until one of a set of file descriptors to become ready to perform I/O. If none of events requested has occured, then poll() blocks until one of the events occurs.

int poll (struct pollfd *fds, nfds_t nfds, int timeout);
-fds: it is a structure that contain the file descriptors (fd), requested events (events) and returned events (revents).
-nfds: it is a array with the speficy number of items.
-timeout: are the millisecond should block waiting for a file descriptor to become ready.

In this function, the return value can be three cases:
-Positive number: the number of structures which have nonzero revents field to struct pollfd.
-Zero: the call timed out.
-(-1): error.

Sources: http://man7.org/linux/man-pages/man2/poll.2.html

Q43 Writing: GPB Verification: JMP

Can we mix file descriptors for sockets and disk files in a single *poll* call?

If you mix socket fd an disk files it will return EFAULT, what means that the array given as argument was not contained in the calling program's address space.

Sources:http://man7.org/linux/man-pages/man2/poll.2.html

Q44 Writing: JMP Verification: GPB

Which events can report a poll call for fds[i].events = POLLIN;?

It report events where there is data to read.

Sources: Sources: http://man7.org/linux/man-pages/man2/poll.2.html

Q45 Writing: GPB Verification: JMP

Are socket options inherited from a listening socket with *accept*?

No, accept function only copies the address family and the protocol from the listening socket.

Sources: http://man7.org/linux/man-pages/man2/accept.2.html

Q46 Writing: JMP Verification: GPB

Which size is the option for SO_RCVLOWAT?

SO_RCVLOWAT is initialized to 1. If the value set is too large, it waits for a smaller one.

Sources: http://man7.org/linux/man-pages/man7/socket.7.html
https://pubs.opengroup.org/onlinepubs/007908799/xns/setsockopt.html

Q47 Writing: JMP Verification: GPB

After poll signals *revents* = POLLOUT on a non-blocking socket, will a very large send block execution?

No when we work with a non-blocking socket get the socket ddoes not block although the size is bigger than the size available of the socket or pipe.

Sources: http://man7.org/linux/man-pages/man2/poll.2.html

Q48 Writing: GPB Verification: JMP

Can *getaddrinfo* return an empty linked list?

It will always return a linked list or 0.

Sources: http://man7.org/linux/man-pages/man3/getaddrinfo.3.html