transfer integer over a socket in C

Ask Question

What is the appropriate way to transfer an `int` over a socket in C? What I am doing so far is:

```
int n = 4;
int tmp = htonl(n);
write(socket, &tmp, sizeof(tmp));
```

and

```
int tmp,n;
read(socket, &tmp, sizeof(tmp));
n = ntohl(tmp);
```

However, the integer received is sometimes 0. Not always, but let's say 2 out of 5 times. It is never some other value, always 0. Why?

UPDATE: Return value from read is -1 and an error is:

```
Resource temporarily unavailable
```

c sockets int

edited Feb 4 '12 at 11:17

asked Feb 4 '12 at 11:08

Maggie
4,183 6 35 60

4 did you check the return value from `read` to see whether it actually filled the buffer? – Alnitak Feb 4 '12 at 11:10

The code snippit you provide is correct. Whatever problem you are having is in code you're not posting. – Brian Roach Feb 4 '12 at 11:12

In case when 0 is received, return from read is -1. When actual integer is received, return from read is 4. This is

I suppose ok, but still - how do I solve it? :/ – Maggie Feb 4 '12 at 11:13

1 Perhaps looking up what that `-1` means from `read` ? It means you didn't read. – Brian Roach Feb 4 '12 at 11:19

2 Did you set `O_NONBLOCK` on the socket? If so, then this error indicates that the data hasn't arrived yet. Try again later. – Steve Jessop Feb 4 '12

at 11:52

3 Answers

First of all, `sizeof(int)` may differ on your sender and receiver machine. So I would recommend you to use something like `int32_t` from `stdint.h` .

Also, it is not guaranteed that `read(..,..,sizeof(int))` will read exactly `sizeof(int)` bytes - it can read nothing, or it can read less bytes. So, the correct variant will be something more like this:

```
int send_int(int num, int fd)
{
    int32_t conv = htonl(num);
    char *data = (char*)&conv;
    int left = sizeof(conv);
    int rc;
    do {
        rc = write(fd, data, left);
        if (rc < 0) {
            if ((errno == EAGAIN) || (
                // use select() or epo
            }
            else if (errno != EINTR) {
                return -1;
            }
        }
        else {
            data += rc;
            left -= rc;
        }
    }
    while (left > 0);
    return 0;
}

int receive_int(int *num, int fd)
{
    int32_t ret;
    char *data = (char*)&ret;
    int left = sizeof(ret);
    int rc;
    do {
        rc = read(fd, data, left);
        if (rc <= 0) { /* instead of r
            if ((errno == EAGAIN) || (
                // use select() or epo
            }
            else if (errno != EINTR) {
                return -1;
            }
        }
        else {
            data += rc;
            left -= rc;
        }
    }
    while (left > 0);
    *num = ntohl(ret);
    return 0;
}
```

edited Oct 30 '17 at 9:00

garlix
313 4 18

answered Feb 4 '12 at 15:57

Borisko
128 4

>UPDATE: Return value from read is -1. Do you use nonblocking IO? – Borisko Feb 4 '12 at 15:58

1 I think that the `receive` function is totally wrong. IMHO you used `ret` instead of `rc` that is actually never used – garlix Sep 30 '15 at 17:06

1 Also you should never look at errno unless read returned -1. You are inspecting it when 0 is returned which is going to lead to a bug that is really painful to track down – D.Shawley Aug 25 '16 at 1:14

1 @garlix is right . Yes! In receive_int(), it should be 'if (rc <= 0) {' instead of 'if (ret <= 0) {'!! – nooooooooob Oct 26 '17 at 16:39

This should work without any problem, try this :

On the sender (Server) side :

```
int number_to_send = 10000; // Put you
int converted_number = htonl(number_to
// Write the number to the opened sock
write(client_socket, &converted_number
```

On the receiver(client) side :

```
int received_int = 0;

return_status = read(client_socket, &r
if (return_status > 0) {
   fprintf(stdout, "Received int = %d\
}
else {
   // Handling erros here
}
```

Hope this will help.

answered Feb 4 '12 at 14:53

iPadDevloperJr
454 2 15 32

1 This should be the accepted answer! Thanks! +1 for `htonl` and `ntohl` –

garlix Sep 30 '15 at 17:07

1 But this does not handle partial reads or writes! read can return the result one byte at a time. – D.Shawley Aug 25 '16 at 1:17

@D.Shawley You're right. This answer has the same problem as the question and solves nothing. You should downvote it for that reason. – Carey Gregory Aug 25 '16 at 1:25

Since no one mentioned `sprintf`

you can just convert any **variable** to `char*` using it and send

```
if(strcmp(buf,"movUP") == 0)
{
    char* msg = calloc(1, 20);
    pos.y += 0.0001f;
    sprintf(msg,"NEW::POS::Y=%.4f", pc
    sendto(master, msg, 20, 0, (struct
}
```

Test

```
movUP
NEW::POS::Y=0.0001
movUP
NEW::POS::Y=0.0002
movUP
NEW::POS::Y=0.0003
movUP
NEW::POS::Y=0.0004
```

Use `%d` for integers, `%f` for floats

to **convert back** to an integer, use `atoi(char*)`
to **convert back** to an float, use `atof(char*)`

before converting, be sure to use `strstr()` to get the float value **only**, starting from `"0"`

```
    float myPos; // floating variable
    ...
    ....
    memset(buf, 0, MAXBUFFER); // clea
    recvfrom(master, buf, MAXBUFFER, 0
    char* newY = strstr(buf, "0");// N
    myPos = atof(newY); // new object0
    printf("My New Position is %.4f\n"
4.4478 -∞ 4.4479 -∞ 4.4470.
```

For integers (**not** positions), you can use the same technique and just multiply it like

```
float f = 0.000002f; // that is suppos
int i = (int)(f*1000000); // now, i ≥
```

the above methods are totally secure

If you want a more solid converting, you can use `strncpy` or `memcpy` and cut the string starting from a given index with some length assuming that you already know the incoming buffer, but in my personal view, I don't really recommend it, specifically in **connectionless** sockets like this one, lots of calculations and calls for buffer length, Not easy to debug sometimes if you're not totally aware of what you're doing.

> **Note 1**: Be careful to **not** include **zeros** in your **buffer** when you're waiting for a server **move**/**position** command or any quantity variable if you're planning to use the **1st** method.
>
> **Note 2**: You can just send your **integer** or **float**, **convert it** and **vice versa** without needing to **cut** or **multiply** it.

May new game networking developers find this answer useful too since we can't send or receive except `char*` with **UDP** sendto(), recvfrom().

edited Mar 9 at 11:37

answered Mar 9 at 2:38

X Stylish
1,370 1 4 29