sign up log in tour help stack overflow careers

Stack Overflow is a question and answer site for professional and enthusiast programmers. It's 100% free, no registration required. Take the 2-minute tour ×

Sending Image (JPEG) through Socket in C Linux

I'm writing a small C program in order to be able to transfer an image file between two computers (from server to client both running linux) using TCP/IP sockets but there seems to be an error as my picture appears on the other sides corrupted.

The code for my server is as such:

```
//Prepare the sockaddr_in structure
server.sin_family = AF_INET;
server.sin_addr.s_addr = INADDR_ANY;
server.sin_port = htons( 8889 );

//Bind
if( bind(socket_desc,(struct sockaddr *)&server , sizeof(server)) < 0)
{
    puts("bind failed");
    return 1;
}

puts("bind done");

//Listen
listen(socket_desc , 3);

//Accept and incoming connection
puts("Waiting for incoming connections...");
c = sizeof(struct sockaddr_in);

if((new_socket = accept(socket_desc, (struct sockaddr *)&client,
(socklen_t*)&c))){
    puts("Connection accepted");
}

fflush(stdout);

if (new_socket<0)
{
    perror("Accept Failed");
    return 1;
}

send_image(new_socket);
```

The client side code which is receiving the data is as such:

```
int main(int argc , char *argv[])
{

int socket_desc;
struct sockaddr_in server;
char *parray;


//Create socket
socket_desc = socket(AF_INET , SOCK_STREAM , 0);

if (socket_desc == -1) {
    printf("Could not create socket");
}

memset(&server,0,sizeof(server));
server.sin_addr.s_addr = inet_addr("10.42.0.1");
server.sin_family = AF_INET;
server.sin_port = htons( 8889 );

//Connect to remote server
if (connect(socket_desc , (struct sockaddr *)&server , sizeof(server)) < 0) {
    cout<<strerror(errno);
    close(socket_desc);
    puts("Connect Error");
    return 1;
}

puts("Connected\n");

receive_image(socket_desc);

close(socket_desc);

return 0;
```

Can anyone give me a hand with this? I can't see to figure out this error for the life of me.

EDIT: I've changed the fwrites and freads back into regular write and read and it still sends a corrupted image

c++ | image | file | sockets | networking

edited Mar 16 '13 at 5:18

asked Mar 16 '13 at 3:23

user1721182

21 1 6

1 You've no need for `ioctl` and `FIONREAD` ; just let it block if there's no data yet. It's not like you're doing anything else in the mean time. – icktoofay Mar 16 '13 at 3:51

Have you confirmed that each individual read_size correponds to each individual write_size? – Dan Nissenbaum Mar 16 '13 at 6:48

add a comment

2 Answers

You have a number of problems:

- You need to open the file in binary mode (`"rb"` for reading, `"wb"` for writing), not the default text mode. On Windows (and any other systems which do line ending translation), the stdio library converts LFs (bytes 0x0A) into CRLF pairs (the two bytes 0x0D 0x0A) when writing, and it does the inverse translation when reading. For non-text data like JPEG files, this corrupts the data.
- There's no need to be sending your "handshake" bytes after each send. TCP/IP already handles acknowledgements/resends/flow control/etc. You can assume that as long as `send()` / `write()` returns a positive value, then that many bytes were received by the other peer.
- `send()` / `write()` may not send all of the data you ask it to—they may do a partial send. If that happens, you need to keep trying to send the rest of the buffer in a loop.
- `sizeof(char)` is guaranteed to be 1 by the C language standard, there's rarely a need to say `sizeof(char)` when instead your code will be much clearer without it
- In the client code, there's no need to use that `ioctl` to determine how much data can be read without blocking because you're just looping again—your code will spin at 100% CPU while there's no data available. Just let the `read()` call block. If you're running this code on a laptop, your battery will thank you.
- Likewise, the client will almost definitely be getting partial reads, you're not going to receive the whole file in a single call. You need to write out whatever data you get, then loop and receive again.
- When you send the image size over the socket at the start, you might get a different value on the client if the two systems are not of the same endianness. In order to make your code bulletproof, you need to convert the data to network order (big-endian) when sending it, then convert it back to host (native) order after receiving it. You can use the `ntohl(3) and htonl(3)` functions to do these conversions for 4-byte values.

answered Mar 16 '13 at 4:30

Adam Rosenfield
187k 42 291 423

I think it's safe to say if you `#include <unistd.h>` , then you don't have to open files with `"rb"` and `"r"` is fine. – Dietrich Epp Mar 16 '13 at 4:44

For the client portion of the code where you mentioned it is getting partial reads, is my code not already taking care of that? I have it reading in as many bytes as are available and then writing to the file the number of bytes returned from the read function and then looping until it has written the number of bytes equal to the file size. – user1721182 Mar 16 '13 at 5:21

Also this image transfer program is only being used for a client and server that will both be running Ubuntu so isn't the opening of the file in binary mode unnecessary? – user1721182 Mar 16 '13 at 5:35

add a comment

Works nicely now.

Best,

Mario.

Client:

```
#include<stdio.h>
#include<string.h>
#include<sys/socket.h>
#include<arpa/inet.h>
#include<sys/ioctl.h>
#include<unistd.h>
#include<iostream>
#include<fstream>
#include<errno.h>
using namespace std;

//This function is to be used once we have confirmed that an image is to be
sent
//It should read and output an image file

int receive_image(int socket)
{ // Start function

int buffersize = 0, recv_size = 0,size = 0, read_size, write_size,
packet_index =1,stat;

char imagearray[10241],verify = '1';
FILE *image;

//Find the size of the image
do{
stat = read(socket, &size, sizeof(int));
}while(stat<0);

printf("Packet received.\n");
printf("Packet size: %i\n",stat);
printf("Image size: %i\n",size);
printf(" \n");

char buffer[] = "Got it";
```

Server:

```
#include<stdio.h>
#include<string.h>
#include<sys/socket.h>
#include<arpa/inet.h>
#include<unistd.h>
#include<iostream>
#include<fstream>
#include<errno.h>

using namespace std;

int send_image(int socket){

FILE *picture;
int size, read_size, stat, packet_index;
char send_buffer[10240], read_buffer[256];
packet_index = 1;

picture = fopen("capture.jpeg", "r");
printf("Getting Picture Size\n");

if(picture == NULL) {
     printf("Error Opening Image File"); }

fseek(picture, 0, SEEK_END);
size = ftell(picture);
fseek(picture, 0, SEEK_SET);
printf("Total Picture size: %i\n",size);

//Send Picture Size
printf("Sending Picture Size\n");
write(socket, (void *)&size, sizeof(int));

//Send Picture as Byte Array
printf("Sending Picture as Byte Array\n");
```

answered Mar 15 at 3:04

mmirand6
1
add a comment

Not the answer you're looking for? Browse other questions tagged c++ image file sockets networking or ask your own question.