Hints for writing the server in C

Java API vs C API

The following methods are found in camera.h. To use the fake version of the camera (using prerecorded images) compile it with the -DFAKE option (preferably using a makefile). As of writing this, excluding the -DFAKE option hasn't yet been implemented in camera.h.

- `public ... TIME_ARRAY_SIZE = 8`
  ⇒ **N/A**
- `public ... IMAGE_BUFFER_SIZE = 128 * 1024`
  ⇒ **`size_t get_frame_size`**
- `public ... IMAGE_WIDTH = 640`
  ⇒ **`size_t get_frame_width(frame* f)`**
- `public ... IMAGE_HEIGHT = 480`
  ⇒ **`size_t get_frame_height(frame* f)`**

- `public void init()`
  ⇒ **N/A**
- `public void setProxy(String host, int port)`
  ⇒ **N/A**
- `public boolean connect()`
  ⇒ **`get_frame_width(frame* f)`**
- `public int getJPEG(byte[] target, int offset)`
  ⇒ **`void* get_frame_bytes(frame* f)`**
- `public boolean motionDetected()`
  ⇒ **`int get_frame_motion(frame* f)`**
- `public void getTime(byte[] target, int offset)`
  ⇒ **`capture_time get_frame_timestamp(frame* f)`** *`//capture_time is of type`* ***`unsigned long long`*** *and the time is measured in nanoseconds*
- `public void close()`
  ⇒ **N/A**
- `public void destroy()`
  ⇒ **`void camera_close(camera* cam)`**

It's also worth noting that the way the java client and c server communicates will be via a socket on an ip address and port, with the server sending bytes and the client receiving them.

Not in Java

- **`void frame_free(frame* f)`** `//To be called on every frame once you're done with it!`

C code hints

Sockets can be found in the #include <sys/socket.h> header. This example has some nice example files showing how to set it up goo.gl/qbSWpd and even run it on the command line.

---

pthread is the thread used in c. You should also use it for mutex locking. Here is a basic example I just quickly found on google that shows how it works:
goo.gl/rHpqsg
Note how a **pthread** takes a me:qthod to run, instead of having a **run()** method, i.e. they're telling the **pthread** to run the method "**doSomething**".
You should also look into the **pthread_cond_broadcast** and **pthread_cond_wait** functions as they may help you.

Potential C problems

Use gdb and valgrind to debug your program. Use -g as an option when compiling to debug your programs. For valgrind, use

```
valgrind ./yourProgram
```

For gdb, use

```
gdb yourProgram
```

Common commands in gdb include

- break [optional file name:]line number // Will pause the program at the line number
- run // Starts the program
- next // Moves one step and *over* function calls
- step // Moves one step and *into* function calls
- continue // Continues execution until the next breakpoint
- backtrace // Shows the current chain of function calls
- list // Shows the surrounding code

The c compiler is sequential, so if it says you haven't defined a method that you actually have defined, it's because you've placed the method below the function in which it is called. If you don't want to move it up, you can just add a definition line before the function in which it is called. Example:

```
void helloWorld();
int main(int argc, char *argv[]){
      helloWorld();
}
void helloWorld(){
```

```
      printf("Hello World!");
}
```

---

size_t is essentially an int that can only be >=0. The number of bits in a size_t (and int) is compiler dependent(!!) and is defined as at least 16 bits.

There are two important types of pointers you'll most likely be using below, and it's important to have at least a basic grasp of the difference. Lets use the example from the bottom.

frame *f // This is a pointer to a struct (essentially an object). If you try to do **f[1]**, the value is // "undefined" and will *probably* give a segmentation fault.

byte *bytes // This is a pointer to the first element in an array (so **bytes[0] == *bytes**). Doing // **bytes[size + 1]** would *probably* give a segmentation fault.

A segmentation fault just says that you're reading or writing to memory that you're not allowed to look at. Compare to Java's ArrayOutOfBoundsException which is a more specific scenario since it is specific to arrays, but in essence means the same.

Example usage

```
#include <stdio.h>
#include <stdlib.h>
#include "camera.h" // Similar to import in java

camera* cam = camera_open();
for (int i = 0; i != 2; ++i) {
      frame *f = camera_get_frame(cam);
      size_t size = get_frame_size(f);
      byte *bytes = get_frame_bytes(f); // bytes is an array of
                                        // size bytes[size], and
                                        // byte is equivalent to
                                         // a char
      for (int j = 0; j != size; ++j)
            printf("%d\n", bytes[j]); // Will print each byte (in
                                      // base 10) on a new line
      frame_free(f);
}
camera_close(cam);
```