CMPT433

How To Guide:

Streaming Webcam from BeagleBone to NodeJS Server

Team Solar

Yoonhong Lee

Jusung Lee

Denys Dziubii

Introduction

In this how-to-guide, we will provide step-by-step instructions on creating a basic web streaming service from a webcam connected to the BeagleBone Green running NodeJS Server. It offers significant improvements (smooth video and low latency) over existing and somewhat outdated guides from 2017 and earlier, so we decided to share our findings from 2022 here. The aim of the guide is to simplify the construction of a video streaming web application with Beaglebone Green.

Required Hardware

- Beaglebone Green
- Logitech c720 Webcam (Any Logitech webcams will work, but c720 is preferred for smoother streaming)

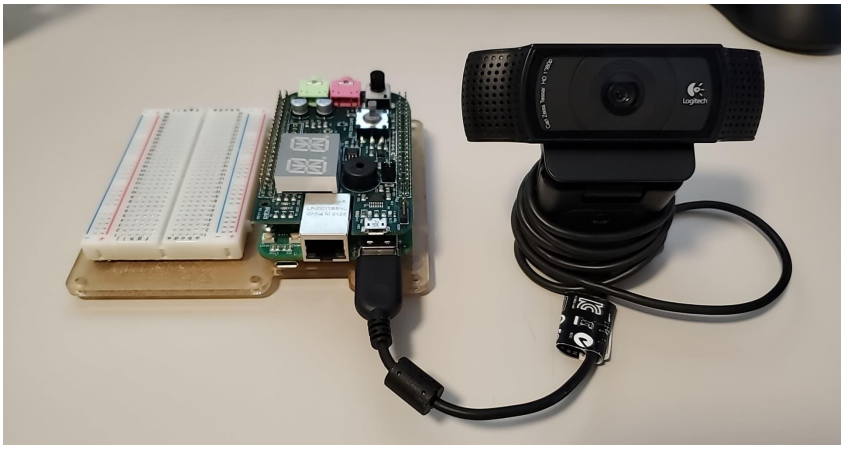
Required Software and Frameworks in the Host

- jQuery v3.6.0
- Node v14.17.3
- Npm v8.5.4
- FFmpeg v5.0.1

1. BeagleBone Green

1.1. Connecting Webcam to the Beaglebone

Connect the webcam to the Beaglebone's USB port next to the ethernet port as shown in the image below.



1.2. Checking the Webcam connection

On the target, enter the following command:

```
debian@yla382-beagle:~$ lsusb
Bus 001 Device 002: ID 046d:082d Logitech, Inc. HD Pro Webcam C920
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

1.3. Modifying capture.c

In this step, we will be downloading derekmolly's capture.c and making modifications to it.

1.3.1. Download capture.c

Go to https://github.com/derekmolloy/boneCV and download capture.c

1.3.2. Modifying capture.c

The original capture.c extracts raw data from the webcam and converts it to the required video extension and then sends its output to STDOUT buffer. We will be making changes by sending its output to NodeJS server using a UDP instead.

1.3.2.1. Adding libraries for UDP sockets

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdbool.h>
#include <arpa/inet.h>
#include <netdb.h>
```

1.3.2.2. Add following the following code

```
#define PORT_T 3000
#define RPORT_T 1234 //Port for NodeJS
static struct sockaddr_in sinT;
static struct sockaddr_in sinRemoteT;
static int socketDescriptorT;

void openConnectionT()
{
	memset(&sinT, 0, sizeof(sinT));
	sinT.sin_family = AF_INET;
	sinT.sin_addr.s_addr = htonl(INADDR_ANY);
	sinT.sin_port = htons(PORT_T);
	socketDescriptorT = socket(PF_INET, SOCK_DGRAM, 0);
	bind(socketDescriptorT, (struct sockaddr*) &sinT, sizeof(sinT));
	sinRemoteT.sin_family = AF_INET;
	sinRemoteT.sin_port = htons(RPORT_T);
	sinRemoteT.sin_addr.s_addr = inet_addr("192.168.7.1");
}

int sendResponseT(const void *str, int size)
{
	int packetSent = 0;
	sendto(socketDescriptorT,
		str,
```

```
                    size,
                    0,
                    (struct sockaddr *) &sinRemoteT,
                    sizeof(sinRemoteT) );
        return packetSent;
}

void closeConnectionT()
{
        close(socketDescriptorT);
}
```

1.3.2.3. Modify static void process_image(cost void *p, int size) (line 131) into the following:

Here, instead of sending data to STDOUT, we will be sending it to the NodeJS server using UDP socket.

```
static void process_image(const void *p, int size)
{
        if (out_buf) {
                sendResponseT(p, size);
        }

        fflush(stderr);
}
```

1.3.2.4. Remove the following code (line 571-606):

The following codes are used for flag settings, but it's not required in our case.

```
static void usage(FILE *fp, int argc, char **argv)
{
        fprintf(fp,
                "Usage: %s [options]\n\n"
                "Version 1.3\n"
                "Options:\n"
                "-d | --device name   Video device name [%s]\n"
                "-h | --help          Print this message\n"
                "-m | --mmap          Use memory mapped buffers [default]\n"
                "-r | --read          Use read() calls\n"
                "-u | --userp         Use application allocated buffers\n"
                "-o | --output        Outputs stream to stdout\n"
                "-f | --format        Force format to 640x480 YUYV\n"
                        "-F | --formatH264    Force format to 1920x1080 H264\n"
                "-c | --count         Number of frames to grab [%i] - use 0 for infinite\n"
                "\n"
                        "Example usage: capture -F -o -c 300 > output.raw\n"
                        "Captures 300 frames of H264 at 1920x1080 - use raw2mpg4 script to convert to
mpg4\n",
                argv[0], dev_name, frame_count);
}
```

```
static const char short_options[] = "d:hmruofFc:";

static const struct option
long_options[] = {
        { "device", required_argument, NULL, 'd' },
        { "help",   no_argument,       NULL, 'h' },
        { "mmap",   no_argument,       NULL, 'm' },
        { "read",   no_argument,       NULL, 'r' },
        { "userp",  no_argument,       NULL, 'u' },
        { "output", no_argument,       NULL, 'o' },
        { "format", no_argument,       NULL, 'f' },
                { "formatH264", no_argument,   NULL, 'F' },
        { "count",  required_argument, NULL, 'c' },
        { 0, 0, 0, 0 }
};
```

1.3.2.5. Modify static void init_device(void) (line 557)

Here, we will be modifying the pixel format, pixel width, and pixel height to the following:

From:

```
if (force_format) {
                        if (force_format==2){
                                fmt.fmt.pix.width       = 1920;
                                fmt.fmt.pix.height      = 1080;
                                fmt.fmt.pix.pixelformat = V4L2_PIX_FMT_H264;
                                fmt.fmt.pix.field       = V4L2_FIELD_INTERLACED;
                        }
                        else if(force_format==1){
                                        fmt.fmt.pix.width   = 640;
                                        fmt.fmt.pix.height  = 480;
                                        fmt.fmt.pix.pixelformat = V4L2_PIX_FMT_YUYV;
                                        fmt.fmt.pix.field       = V4L2_FIELD_INTERLACED;
                        }

                if (-1 == xioctl(fd, VIDIOC_S_FMT, &fmt))
                        errno_exit("VIDIOC_S_FMT");

                /* Note VIDIOC_S_FMT may change width and height. */
        } else {
                /* Preserve original settings as set by v4l2-ctl for example */
                if (-1 == xioctl(fd, VIDIOC_G_FMT, &fmt))
                        errno_exit("VIDIOC_G_FMT");
        }
```

To: (You may change the fmt.fmt.pix.width and fmt.fmt.pix.width to improve the video quality, but it may slow down the video due to hardware limitations)

```
if (force_format) {
                        if (force_format==2){
                            fmt.fmt.pix.width       = 720;
                            fmt.fmt.pix.height      = 720;
                            fmt.fmt.pix.pixelformat = V4L2_PIX_FMT_MJPEG;
                            fmt.fmt.pix.field = V4L2_FIELD_NONE;
                        }
                        else if(force_format==1){
                                    fmt.fmt.pix.width    = 640;
                                    fmt.fmt.pix.height  = 480;
                                    fmt.fmt.pix.pixelformat = V4L2_PIX_FMT_MJPEG;
                    fmt.fmt.pix.field = V4L2_FIELD_NONE;
                            }

            if (-1 == xioctl(fd, VIDIOC_S_FMT, &fmt))
                    errno_exit("VIDIOC_S_FMT");

            /* Note VIDIOC_S_FMT may change width and height. */
      } else {
            /* Preserve original settings as set by v4l2-ctl for example */
            if (-1 == xioctl(fd, VIDIOC_G_FMT, &fmt))
                    errno_exit("VIDIOC_G_FMT");
      }
```

1.3.2.6. Modify Main function

Change the main function to the following:

```
int main()
{
      printf("Starting streaming\n");
      openConnectionT();
      dev_name = "/dev/video0";

      force_format=2;
      out_buf++;

      // errno = 0;
      char opt = '0';
      frame_count = strtol(&opt, NULL, 0);
      if(errno) errno_exit(&opt);

      open_device();
      init_device();
      start_capturing();
      mainloop();
      stop_capturing();
      uninit_device();
      close_device();
      fprintf(stderr, "\n");
      closeConnectionT();
      printf("Ending streaming\n");
      return 0;
}
```

1.4. Create makefile to create executable file called “capture”

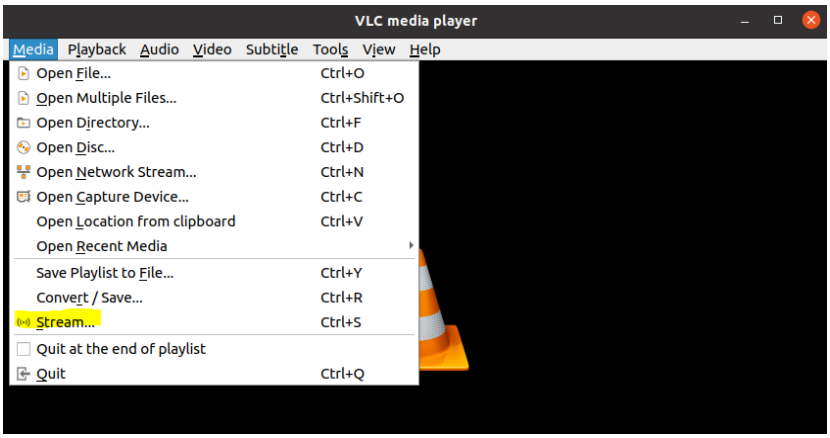
1.5. Testing the streaming through VLC media player

1.5.1. Run the capture script

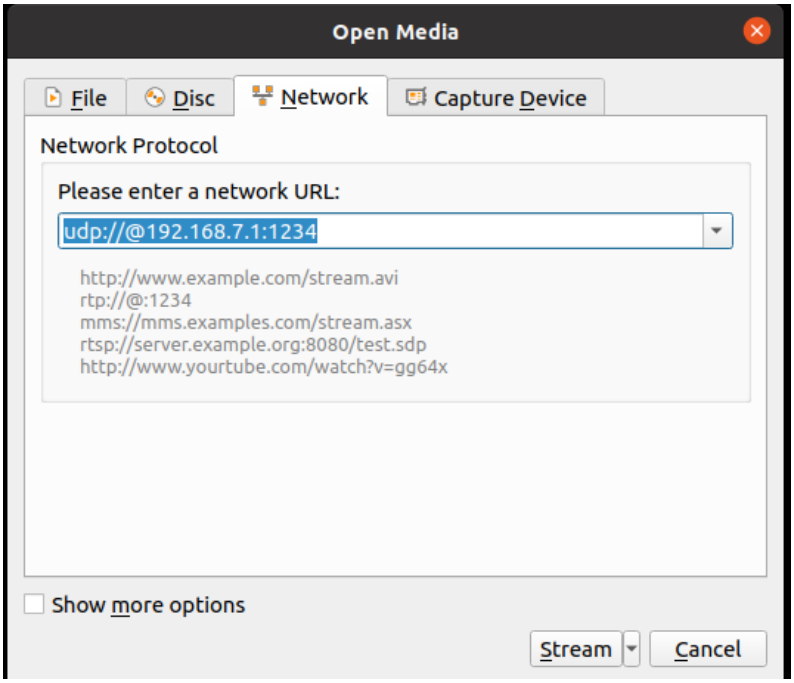
```
debian@yla382-beagle:/mnt/remote/myApps$ ./capture
Starting streaming
Force Format 2
```

1.5.2. Open VLC player

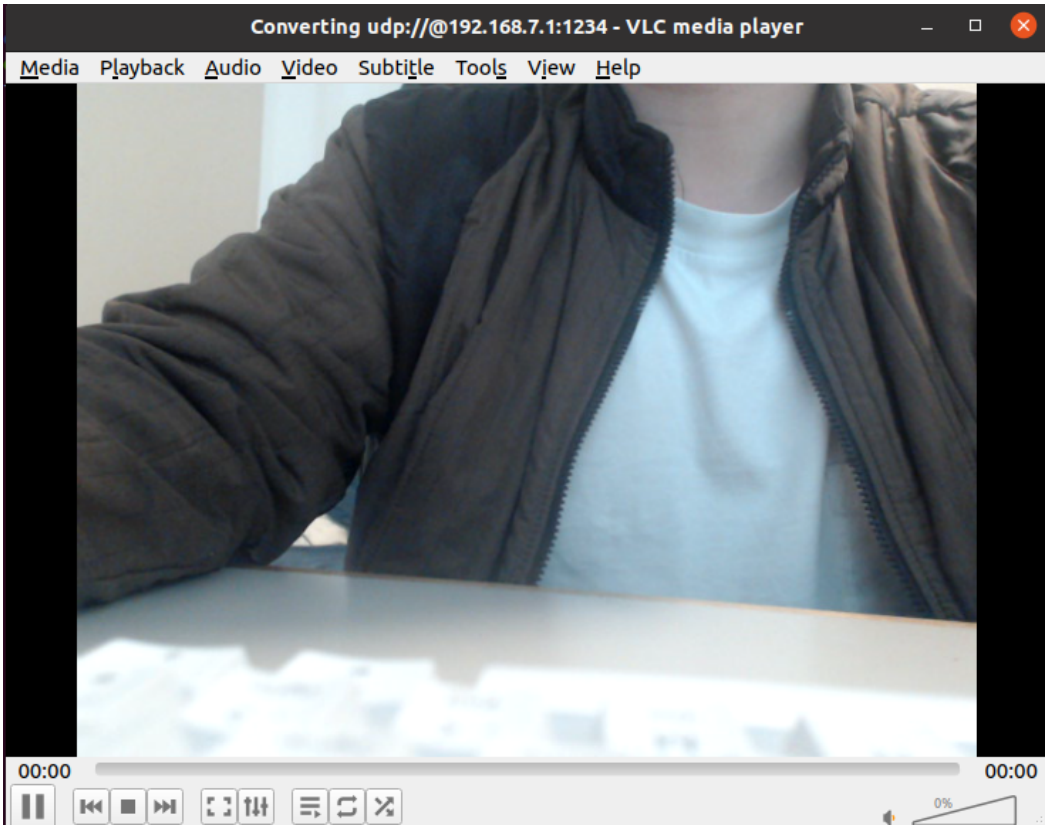
1.5.3. Go to “Media” and select “Stream”



1.5.4. Go to “Network” Setting and enter “udp://@192.168.7.1:1234” then click “Stream”



1.5.5. Click “Stream”



2. NodeJS

2.1. Before you start

2.1.1. Install FFMPEG on host

```
debian@yla382-beagle:~$ sudo apt update
debian@yla382-beagle:~$ sudo apt install ffmpeg
```

2.1.2. Confirm its installed successfully

```
debian@yla382-beagle:~$ ffmpeg -version
```

2.2. Project Structure

— index.js
— package.json
— routers
    — page.js
— public
    — homepage.html
    — script.js

2.3. Run “npm init”

```
yoon@ubuntu:~/cmpt433/work/how-to-guide$ npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help init` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (how-to-guide)
version: (1.0.0)
description:
entry point: (index.js)
test command:
git repository:
keywords:
author:
license: (ISC)
About to write to /home/yoon/cmpt433/work/how-to-guide/package.json:

{
  "name": "how-to-guide",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
Is this OK? (yes) yes
```

2.4. Update the dependencies in package.json and run “npm install”

```
"dependencies": {
    "express": "^4.17.3",
    "socket.io": "^4.4.1",
  }
```

2.5. Create index.js

In index.js, initialize socket.io connection and create a child process to receive UDP packets from capture.c:

```
const express = require('express');
const app = express();
const http = require('http');
const server = http.createServer(app);
const { Server } = require("socket.io");
const io = new Server(server);
const startRouter = require('./routers/page.js');
const {SERVER_PORT: port = 3000} = process.env;
const child = require('child_process');

app.use('/', startRouter);

io.on('connection', (socket) => {
	console.log('a user connected');

	let ffmpeg = child.spawn("ffmpeg", [
		"-re",
		"-y",
		"-i",
		"udp://192.168.7.1:1234",
		"-preset",
		"ultrafast",
		"-f",
		"mjpeg",
		"pipe:1"
		]);

	ffmpeg.on('error', function (err) {
		console.log(err);
		throw err;
	});

	ffmpeg.on('close', function (code) {
		console.log('ffmpeg exited with code ' + code);
	});

	ffmpeg.stderr.on('data', function(data) {
		// Don't remove this
		// Child Process hangs when stderr exceed certain memory
	});

	ffmpeg.stdout.on('data', function (data) {
		var frame = Buffer.from(data).toString('base64'); //convert raw data to string
		io.sockets.emit('canvas',frame); //send data to client
	});
});

server.listen({ port }, () => {
	console.log(`🚀 Server ready at http://0.0.0.0:${port}`);
});
```

2.6. Implement page.js

This script handles request for HTML and JS files:

```
const express = require('express');
const router = express.Router();
const path = require('path');
const filePath = "/../public/";

router.get('/', function(req, res) {
        res.sendFile(path.join(__dirname + filePath + "homepage.html"));
});

router.get('/script.js', function(req, res) {
        res.sendFile(path.join(__dirname + filePath + "script.js"));
});

module.exports = router;
```

2.7. Implement homepage.html

Create a simple html file containing canvas tage where video will be displayed. Ensure that canvas width and height are the same as capture.c's video size setting:

```
<!DOCTYPE html>
<html>
   <head>
      <title>Video Streaming</title>
      <meta charset="utf-8">
      <meta name="viewport" content="width=device-width, initial-scale=1">
   </head>
   <body>

       <canvas id="videostream" width="720" height="720"></canvas>


      <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
      <script src="/socket.io/socket.io.js"></script>
      <script src="script.js"></script>
   </body>
</html>
```

2.8. Implement script.js

Whenever the NodeJS server sends a new video frame, convert it to an image and replace the existing image in the canvas tag

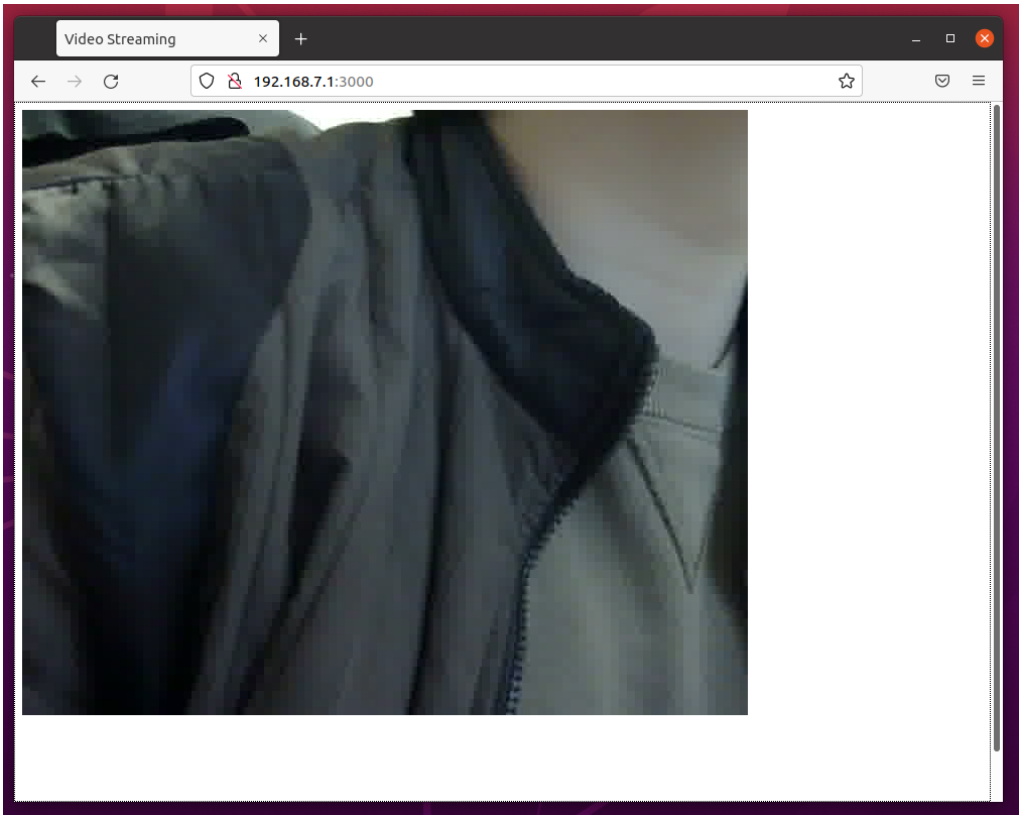
```
const socket = io();
socket.on("connect", (socket) => { //confirm connection with NodeJS server
   console.log("Connected");
});

$( document ).ready(function() {
   socket.on('canvas', function(data) {
      const canvas = $("#videostream");
      const context = canvas[0].getContext('2d');
      const image = new Image();
      image.src = ""+data;
      image.onload = function(){
         context.height = image.height;
         context.width = image.width;
         context.drawImage(image,0,0,context.width, context.height);
      }
```

```
    });
});
```

2.9. Run capture.c on the target and run “npm start” on the host

Go to 192.168.7.1:3000 in the host browser



3. Troubleshoot

3.1. Unable to re-run capture.c

The capture.c runs using an infinite loop. To close the program, force quit will need to be used. When that happens, port required for the script will still be unavailable due to the program not being safely terminated.

3.1.1. Manually kill process running capture.c

3.1.1.1. Check currently running processes

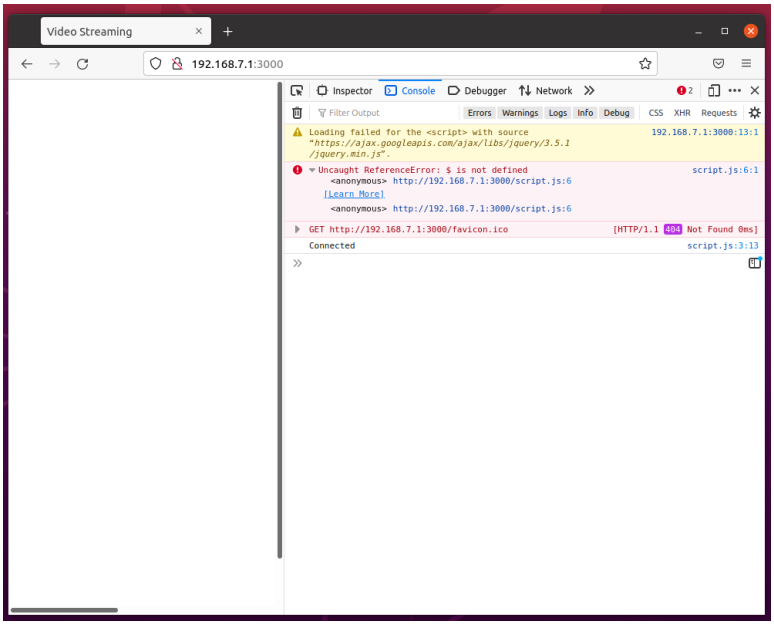
```
debian@yla382-beagle:~$ ps -aux
USER       PID  %CPU %MEM    VSZ   RSS   TTY      STAT   START   TIME  COMMAND
…
debian    1003     3.3     0.9   5120  4664  pts/0      T     21:12   0:32       ./capture
root      1004     0.0     0.0      0     0      ?      S     21:16   0:00  [kworker/0:1]
debian    1011     0.0     0.4   6728  2456  pts/0      R+    21:28   0:00       ps -aux
```

3.1.1.2. Kill the process using ./capture

```
debian@yla382-beagle:~$ kill -9 1003
[1]+  Killed                  ./capture
```

3.1.2. Restart VM

3.2. Unable to load jQuery 4after refreshing (F5)



Due to Firefox's default settings in about:config, the browser will not be able to load jQuery CDN.

3.2.1. Use Chrome

3.2.2. Use downloaded jQuery instead of jQuery CDN

References

[1] https://opencoursehub.cs.sfu.ca/bfraser/grav-cms/cmpt433/links/files/2017-student-howtos/CapturingAndStreamingWebcamVideoOnBBG.pdf

[2] https://opencoursehub.cs.sfu.ca/bfraser/grav-cms/cmpt433/links/files/2018-student-howtos/GrovePIRMotionSensorAndLiveStreaming.pdf

[3] https://github.com/geraldo/nodeStream/tree/master/server

[4] https://github.com/derekmolloy/boneCV