Raspberry Pi Controller

Ashton Warner

November 2021

Contents

1 Introduction

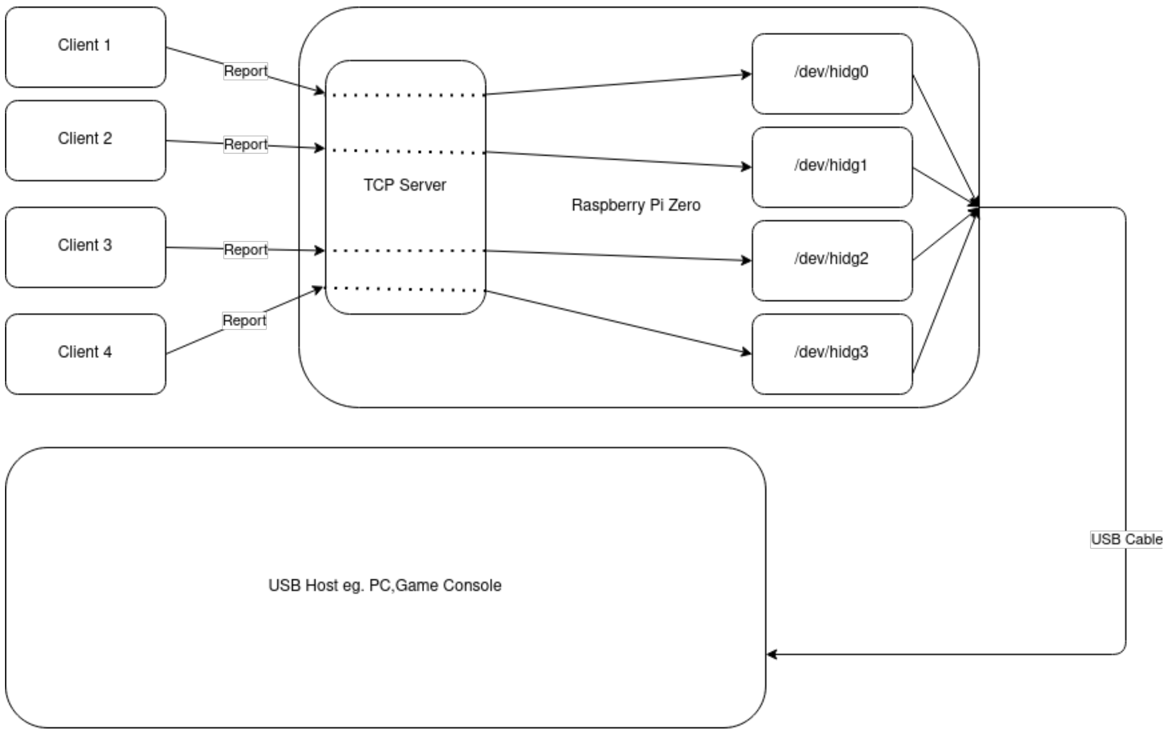
The Rapsberry Pi Controller was a project that I was inspired by Joycon drift. My brother and I loved playing on the Nintendo Switch but the Joycon drift made the experience annoying. One day we found a python script that allowed you to connect to the Nintendo Switch via bluetooth, but it had one fault. The bluetooth was glitchy and had issues connecting reliably. Knowing that the Raspberry Pi could be used as a USB HID device I started researching how to turn my Raspberry Pi into a Fake PDP. Faceoff Wired Pro Controller. I found a tutorial on isticktoit.net [1] for turning the Raspberry Pi Zero into a USB gadget and I proceeded worked on many versions before I stumbled upon how to turn it into a functional controller. But I knew the Nintendo switch could connect four controllers at the same time, so I decided to try and discover how to use the libcomposite Kernel Module to make it act as several controllers. I managed to do this and then debugged for several days before writing a TCP server to allow Keyboard and any other HID device your computer, or any other network device, can recognise to send an USB HID report to send it to the server.

2 About

2.1 Description

The raspberry pi zero controller works from the kernel module libcomposite which allows the raspberry pi to act as a USB device. You must use a raspberry pi zero or raspberry pi 4 because only the raspberry pi zero and raspberry pi 4 have direct access to the USB port which allows them to pretend to be a USB device. I created a small bash script that configured the USB port as a hub with access to 4 virtual PDP. Faceoff Wired Pro Controllers. The script created 4 files (/dev/hidgX) with byte write capability. I created a TCP server in C to allow forwarding USB HID reports to the/dev/hidgX device files.

2.2 Diagram



3 Usage

3.1 Report Format

3.1.1 Bytes

AA BB CC DD EE FF GG HH
Bit Order - 76543210

3.1.2 AA Buttons 0-7 - 1 button per bit

0000 0000
ZR ZL R L X A B Y

3.1.3 BB Buttons 8-13 - 1 button per bit - bits 6-7 stay at 0

0000 0000
0 0 Capture Home RightStickClick LeftStickClick Plus Minus

3.1.4 CC Dpad - Compass Directions - bits 5-7 stay at 0

N	0x00
NE	0x01
E	0x02
SE	0x03
S	0x04
SW	0x05
W	0x06
NW	0x07
Centre	0x0f

3.1.5 DD-GG Stick Axis

DD - Left Stick X
EE - Left Stick Y
FF - Right Stick X
GG - Right Stick Y

Min	0x00
Centre	0x7f
Max	0xff

3.1.6 HH NULL Byte

HH - always stays at 0x00

4 Report Descriptor

```
0x05, 0x01,       // Usage Page (Generic Desktop Ctrls)
0x09, 0x05,       // Usage (Game Pad)
0xA1, 0x01,       // Collection (Application)
0x15, 0x00,       // Logical Minimum (0)
0x25, 0x01,       // Logical Maximum (1)
0x35, 0x00,       // Physical Minimum (0)
0x45, 0x01,       // Physical Maximum (1)
0x75, 0x01,       // Report Size (1)
0x95, 0x0E,       // Report Count (14)
0x05, 0x09,       // Usage Page (Button)
0x19, 0x01,       // Usage Minimum (0x01)
0x29, 0x0E,       // Usage Maximum (0x0E)
0x81, 0x02,       // Input (Data,Var,Abs,No Wrap,Linear,Preferred State,No Null Position)
0x95, 0x02,       // Report Count (2)
0x81, 0x01,       // Input (Const,Array,Abs,No Wrap,Linear,Preferred State,No Null Position)
0x05, 0x01,       // Usage Page (Generic Desktop Ctrls)
0x25, 0x07,       // Logical Maximum (7)
0x46, 0x3B, 0x01, // Physical Maximum (315)
0x75, 0x04,       // Report Size (4)
0x95, 0x01,       // Report Count (1)
0x65, 0x14,       // Unit (System: English Rotation, Length: Centimeter)
0x09, 0x39,       // Usage (Hat switch)
0x81, 0x42,       // Input (Data,Var,Abs,No Wrap,Linear,Preferred State,Null State)
0x65, 0x00,       // Unit (None)
0x95, 0x01,       // Report Count (1)
0x81, 0x01,       // Input (Const,Array,Abs,No Wrap,Linear,Preferred State,No Null Position)
0x26, 0xFF, 0x00, // Logical Maximum (255)
0x46, 0xFF, 0x00, // Physical Maximum (255)
0x09, 0x30,       // Usage (X)
0x09, 0x31,       // Usage (Y)
0x09, 0x32,       // Usage (Z)
0x09, 0x35,       // Usage (Rz)
0x75, 0x08,       // Report Size (8)
0x95, 0x04,       // Report Count (4)
0x81, 0x02,       // Input (Data,Var,Abs,No Wrap,Linear,Preferred State,No Null Position)
0x75, 0x08,       // Report Size (8)
0x95, 0x01,       // Report Count (1)
0x81, 0x01,       // Input (Const,Array,Abs,No Wrap,Linear,Preferred State,No Null Position)
0xC0,             // End Collection
```

5 Scripts And Files

5.1 USB Device Creation

Listing 1: bash version

```bash
#!/bin/bash
modprobe libcomposite
cd /sys/kernel/config/usb_gadget/
mkdir -p g0
cd g0
```

```
echo 0x0e6f > idVendor # Linux Foundation
echo 0x0180 > idProduct # Multifunction Composite Gadget
echo 0x0572 > bcdDevice # v1.0.0
echo 0x0200 > bcdUSB # USB2
echo 0x00   > bDeviceClass
echo 0x00   > bDeviceSubClass
echo 0x00   > bDeviceProtocol
mkdir -p strings/0x409
echo "fedcbabcdef" > strings/0x409/serialnumber
echo "PDP CO.,LTD." > strings/0x409/manufacturer
echo "Faceoff Wired Pro Controller for Nintendo Switch" >
strings/0x409/product

create_function() {
  mkdir -p functions/hid.usb$1
  echo 0 > functions/hid.usb$1/protocol
  echo 0 > functions/hid.usb$1/subclass
  echo 64 > functions/hid.usb$1/report_length # 8 bytes. 8 bits per byte
  echo -ne  \\x05\\x01\\x09\\x05\\xA1\\x01\\x15\\x00\\x25\\x01\\x35\\x00
  \\x45\\x01\\x75\\x01\\x95\\x0E\\x05\\x09\\x19\\x01\\x29\\x0E\\x81\\x02
  \\x95\\x02\\x81\\x01\\x05\\x01\\x25\\x07\\x46\\x3B\\x01\\x75\\x04\\x95
  \\x01\\x65\\x14\\x09\\x39\\x81\\x42\\x65\\x00\\x95\\x01\\x81\\x01\\x26
  \\xFF\\x00\\x46\\xFF\\x00\\x09\\x30\\x09\\x31\\x09\\x32\\x09\\x35\\x75
  \\x08\\x95\\x04\\x81\\x02\\x75\\x08\\x95\\x01\\x81\\x01\\xC0 >
  functions/hid.usb$1/report_desc
  # report descriptor from PDP CO.,LTD Faceoff Wired Pro Controller for
  # Nintendo Switch
}

create_function 0 # /dev/hidg0
create_function 1 # /dev/hidg1
create_function 2 # /dev/hidg2
create_function 3 # /dev/hidg3

create_config() {
  mkdir -p configs/c.$1/strings/0x409
  echo "Gamepad Configuration" > configs/c.$1/strings/0x409/configuration
  echo 0x80 > configs/c.$1/bmAttributes
  echo 500 > configs/c.$1/MaxPower
}

link_config() {
    ln -s functions/hid.usb$2 configs/c.$1/
}

create_config 1 # create USB port 1 config

link_config 1 0 # link /dev/hidg0 to USB port 1
link_config 1 1 # link /dev/hidg1 to USB port 1
link_config 1 2 # link /dev/hidg2 to USB port 1
link_config 1 3 # link /dev/hidg3 to USB port 1

sudo ls /sys/class/udc > UDC # Setup USB gadget to USB OTG port
```

```
sleep 10 # Ensures that it is set up before we give it data
```

5.2 TCP Server

Listing 2: c version

```c
#include <stdio.h>
#include <stdbool.h>
#include <netdb.h>
#include <netinet/in.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>
#include <pthread.h>
#include <arpa/inet.h>
#include <fcntl.h>

#define MAX 8 // Report Descriptor Length in Bytes
#define PORT 8080 // TCP Port
#define SA struct sockaddr

typedef struct {
  int sockfd;
  int id;
  char file[12];
} data_t; // allows us to pass data easily to functions

pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;
unsigned char empty[8] = {
  0x00, 0x00, 0x08, 0x80, 0x80, 0x80, 0x80, 0x80
};
// Controller With released Buttons, Dpad, and Sticks

// Writes Report to /dev/hidgX
void send_report(unsigned char* data, int fd) {
  write(fd, data, 8);
  fdatasync(fd);
}

// Function designed for chat between client and server.
// Used in a thread
void * func(void *arg)
{
  // Contains file info, TCP Socket info, and Client ID
  data_t* args = (data_t *)(arg);
  printf("Client %d\n", args->id);
  unsigned char buf[MAX]; // Report Descriptor of 8 bytes

  // Open and prepare /dev/hidgX
  int fd = open(args->file, O_RDWR);
  fd_set rfds;
```

```
FD_ZERO(&rfds);
FD_SET(fd, &rfds);


// forever loop for chat
for (;;) {
  // Recieve Report Descriptor
  read(args->sockfd, buf, 8);

  //Lock pthread to prevent multi-access
  pthread_mutex_lock(&lock);

  bool sbreak = true;

  int i;
  for(i = 0; i < 8; i++) {
    // Is the client about to close the socket
    // close report descriptor = 0xFFFFFFFFFFFFFFFF
    // If there is a none 0xff byte we are still receiving
    if(buf[i] != 0xff) {
      sbreak = false;
      break;
    }
  }

  // If the client is going to close the socket
  // unlock pthread so the other threads can run
  // and exit forever loop
  if(sbreak) {
    pthread_mutex_unlock(&lock);
    break;
  }

  // push report_descriptor to /dev/hidgX
  send_report(buf, fd);

  // Allow other threads to run
  pthread_mutex_unlock(&lock);
}

// Close socket with client
close(args->sockfd);

// Reset Controller for USB Host
send_report(empty, fd);

// Close /dev/hidgX
close(fd);

// Allow a new Client to use this controller
args->sockfd = -1;

// Exit the Thread
pthread_exit(NULL);
```

```
}

// Main function
int main()
{
  int sockfd, connfd, len;
  data_t data[4];

  // Allow all data sockets to be assigned
  data[0].sockfd = -1;
  data[1].sockfd = -1;
  data[2].sockfd = -1;
  data[3].sockfd = -1;

  // Keep track of all socket threads with the clients
  pthread_t clients[4];
  struct sockaddr_in servaddr, cli;
  int optval;

  // socket create and verification
  sockfd = socket(AF_INET, SOCK_STREAM, 0);
  if (sockfd == -1) {
    printf("socket creation failed...\n");
    exit(0);
  }
  optval = 1;
  setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR,
             (const void *)&optval , sizeof(int));
  bzero(&servaddr, sizeof(servaddr));

  // assign IP, PORT
  servaddr.sin_family = AF_INET;
  servaddr.sin_addr.s_addr = inet_addr("0.0.0.0");
  servaddr.sin_port = htons(PORT);

  // Binding newly created socket to given IP and verification
  if ((bind(sockfd, (SA*)&servaddr, sizeof(servaddr))) != 0) {
    printf("socket bind failed...\n");
    exit(0);
  }

  // Now server is ready to listen and verification
  if ((listen(sockfd, 4)) != 0) {
    printf("Listen failed...\n");
    exit(0);
  }
  len = sizeof(cli);

  int id;
  // Accept the data packet from client and verification
  while(1) {
    connfd = accept(sockfd, (SA*)&cli, &len);
    if (connfd < 0) {
      printf("server accept failed...\n");
      exit(0);
    }

    char full = 1;
    for(id = 0; id < 4; id++) {
      // Make sure not to overwrite a clients socket
      if(data[id].sockfd == -1) {
        full = 0;
        data[id].sockfd = connfd;
        data[id].id = id;
        sprintf(data[id].file, "/dev/hidg%d", id);
        pthread_create(&clients[id], NULL, func, &data[id]);
        break;
      }
    }

    // If there a no more client spaces
    // Close Connection with incoming Client
    if(full) {
      close(connfd);
    }
  }

  for(id = 0; id < 4; id++) {
    pthread_join(clients[id], NULL);
  }
  // Close Server Socket
  close(sockfd);
  return 0;
}
```

5.3 rc.local

Listing 3: bash version

```
...
sudo su

# Enable USB HID on the USB OTG port
RUN_HID_SCRIPT

# Enable Complete Access to the /dev/hidgX files
sudo chmod 777 /dev/hidg0
sudo chmod 777 /dev/hidg1
sudo chmod 777 /dev/hidg2
sudo chmod 777 /dev/hidg3

# Run the TCP server in the background
RUN_TCP_SERVER &
...
```

6 Bibliography

References

[1] Composite USB Gadgets on the Raspberry Pi Zero, isticktoit.net

[2] Files based off https://github.com/aidantwoods/RPi0w-keyboard

[3] My Customised Files https://github.com/drflamemontgomery/RaspberryPiZero-Controller