Xbox 360 controller demo

Student:	A.W. Janisse
Studentnummer:	2213829
Instelling:	Fontys hogescholen te Eindhoven
Opleiding:	ICT & Technology
Docent:	Dhr. Erik Dortmans

Opdracht:	Eindopdracht
Datum:	26 juni 2015

Inhoud

Overzicht listings

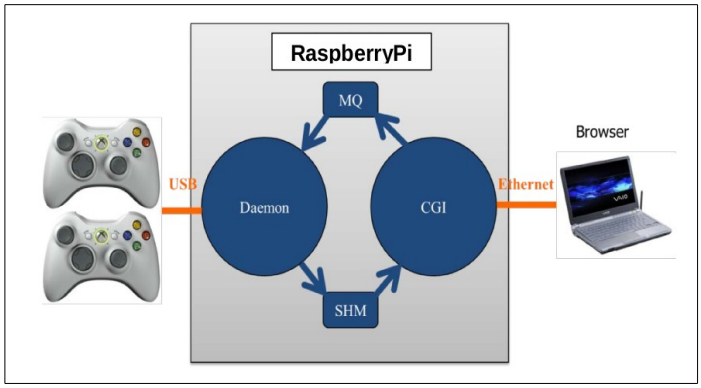
Overzicht afbeeldingen

1 Inleiding

In deze opdracht wordt een complete embedded webservice applicatievoor het uitlezen en besturen van gamepads gerealiseerd.

1.1 Taken

Bouw het gamepad besturingsprogramma wat je in een eerdere opdracht hebt gerealiseerd om tot een daemon – een service – die op de achtergrond draait. Daarnaast dient een losse CGI-applicatie geschreven te worden, die vanuit een remote browser via het netwerk kan worden aangeroepen, en die door middel van IPC mechanismes, met name POSIX MessageQueue en SharedMemory, communiceert met deze gamepad daemon. Het totale systeem de structuur hebben zoals afgebeeld in afbeelding 1.



Afbeelding 1: Structuur van het systeem

1.2 Beschrijving van de onderdelen

Gamepad
De Xbox 360 controllers worden aangesloten op de USB poorten van de Raspberry Pi. In totaal kunnen er vier controllers worden aangesloten.

Deamon
De Deamon is een C applicatie die automatisch wordt opgestart als de Raspberry Pi wordt opgestart. Deze applicatie is verantwoordelijk voor het inlezen van controller knoppen en het aansturen van de leds of de rumble motor.

CGI
De CGI applicatie wordt aangeroepen door de website. De applicatie is verantwoordelijk voor het ontvangen en versturen van gegevens van en naar de website.

MQ
De message queue (MQ) is een mechanisme in het operating system dat het mogelijk maakt om berichten van en naar verschillende processen te versturen. Zo worden commando's die vanuit de website worden gegeven als een bericht doorgestuurd naar de Deamon applicatie.

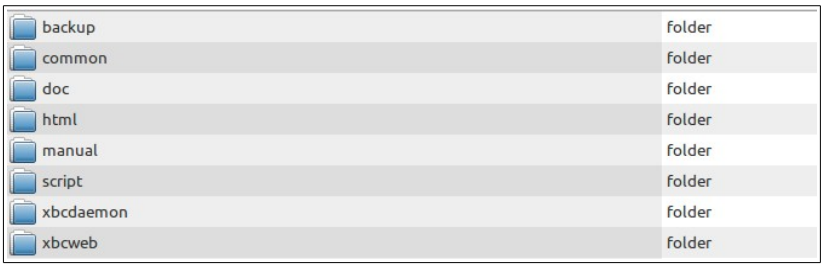
SHM
De shared memory (SHM) is een file die gezamenlijk door de Deamon en de CGI applicatie worden gebruikt. De Deamon schrijft van alle aangesloten controllers de status van de knoppen in deze file. De CGI applicatie leest uit het SHM de knoppen status en stuurt deze vervolgens door naar de website.

Website
De website is een HTML pagina die opgeroepen kan worden met een internetbrowser. De website visualiseert de controllers en vanuit hier kunnen commando's gegeven worden om van iedere afzonderlijke controller de leds en/of de rumble motor aan of uit te zetten.

***Raspberry* Pi**
De Raspberry Pi is een microcontroller board waar het Linux operating systeem draait. Dit bord bezit over een aantal aansluitingen waaronder USB poorten en een ethernet poort. Dit zijn de poorten die gebruikt worden in deze applicatie.

2 Project mappen

Het project heeft de volgende mappen structuur:



Afbeelding 2: Project mappen structuur

2.1 Beschrijving van de mappen

backup: Locatie waar de back-ups worden geplaatst zie met Make gemaakt zijn.

common: Gemeenschappelijke broncode files voor de Deamon en CGI applicatie.

doc: Broncode documentatie in HTML formaat.

***html*:** De HTML website

manual: Beschrijving van de werking

***script*:** Shell script war er voor zorgt dat de Deamon automatisch opstart.

***xbcdeamon*:** Hierin staan alle broncode files die nodig zijn voor de Deamon applicatie.

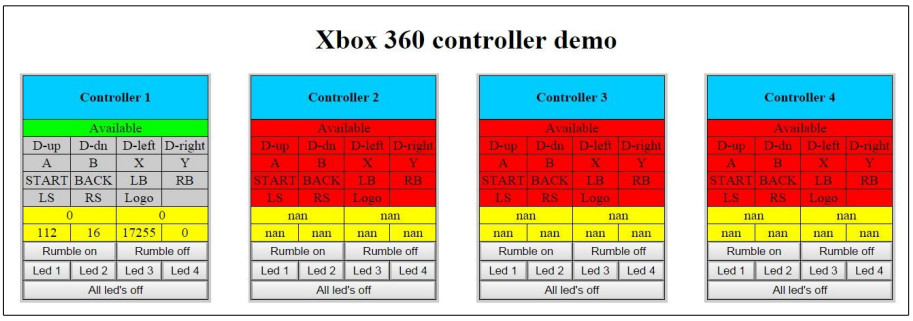
***xbcweb*:** Hierin staan alle broncode files die nodig zijn voor de CGI applicatie.

3 De website

Naam: xbc.html
Builden: Niet van toepassing.
Installatie: Kopieer de file op de Raspberry Pi in: /var/www
(Bijvoorbeel: $ scp xbc.html root@10.0.0.42:/var/www)

3.1 De interface

De website is eenvoudig en maakt gebruik van zowel JavaScript als HTML. De site zie er na openen, met een aangesloten controller, uit zoals te zien is in afbeelding 3. Zodra een controller wordt aangesloten op de Raspberry Pi zal het vlak 'Available' groen oplichten en zullen de andere vlakken de actuele status van knoppen en joy-sticks laten zien.



Afbeelding 3: Interface website

3.2 JavaScript

Het hart van de site wordt gevormd door een aantal functies in JavaScript. Als de website wordt opgevraagd dan wordt functie getbuttons aangeroepen. Zie listing 1. Vanuit hier wordt met een 'get' de CGI applicatie opgeroepen. De oproep heeft het volgende vast formaat:

xbcweb?<id>,<cmd>,<value>

De parameters hierbij zijn: <id> het controller nummer (0 t/m 3), <cmd> het commando 'btns' en <value> heeft hierbij geen betekenis.

```
function getButtons()
{
    var req = null;

    if(window.XMLHttpRequest)
        req = new XMLHttpRequest();
    else if (window.ActiveXObject)
        req  = new ActiveXObject(Microsoft.XMLHTTP);

    req.onreadystatechange = function()
    {
        if(req.readyState == 4)
        {
            if(req.status == 200)
            {
                updateController(idx, req.responseText);
                next();
                getButtons();
            }
            else
            {
                // TODO: report error;
            }
        }
    };

    req.open("GET", "/cgi-bin/xbcweb?" + idx + ",btns,0", true);
    req.send(null);
}
```

Listing 1: functie getButtons

Als na het versturen van met 'get' vanuit deze functie een antwoord wordt gekregen dan wordt de functie updatecontroller aangeroepen. Vanuit deze functie wordt bepaald bij welke controller de ontvangen data hoort en zal de betreffende controller op het scherm aanpassen. Zie listing 2.

```
function updateController(id, data)
{
    var elem = null;
    var b = data.split(",");

    switch(id) {
        case 0: updateControllerElements(id, b);
                break;
        case 1: updateControllerElements(id, b);
                break;
        case 2: updateControllerElements(id, b);
                break;
        case 3: updateControllerElements(id, b);
                break;
    }
}
```

Listing 2: functie updateController

Het updaten van de website elementen gebeurt in functie updateControllerElements. Listing 3 laat een gedeelte van deze functie zien. In deze functie worden de elementen van de betreffende controller op het scherm geüpdatet. Zoals te zien is worden de elementen geselecteerd met hun naam en het meegegeven nummer (id), bijvoorbeeld 'AVAIL_0'.

```
function updateControllerElements(id, b)
{
    elem = document.getElementById('AVAIL_'+id);
    updateBoolElem(elem, b[AVAIL]);
    elem = document.getElementById('D_UP_'+id);
    updateBoolElem(elem, b[D_UP]);
    elem = document.getElementById('D_DN_'+id);
    updateBoolElem(elem, b[D_DN]);
    elem = document.getElementById('D_LEFT_'+id);
    updateBoolElem(elem, b[D_LEFT]);
    elem = document.getElementById('D_RIGHT_'+id);
    updateBoolElem(elem, b[D_RIGHT]);

    elem = document.getElementById('A_'+id);
```

Listing 3: functie updateControllerElements

Het aansturen van de leds en de rumble motor per controller is mogelijk met de knoppen op de website. Als een van deze knoppen wordt bediend dan wordt de functie sendCommand aangeroepen. Listing 4 laat deze functie zien. Vanuit deze functie wordt met een 'get' data naar de CGI applicatie gestuurd en heeft de volgende structuur:

xbcweb?<id>,<cmd>,<value>

Hierin is <id> het controller nummer (0 t/m 3), <cmd> het commando welke 'led' of 'rumble' kan zijn. <value> geeft bij het 'led' commando een van van functies aan en bij 'rumble' de snelheid van de rumble motor. De rumble snelheid kan variëren van 0 t/m 0xFFFF. De led functie worden hierna in paragraaf 3.2.1 beschreven.

```
function sendCommand(id,command,value)
{
    var xmlHttp = new XMLHttpRequest();
    xmlHttp.open("GET", "/cgi-bin/xbcweb?" + id + "," + command + "," + value, true);
    xmlHttp.send( null );
}
```

Listing 4: functie sendCommand

3.2.1 Overzicht led functies van de controller:

0	Alle leds uit
1	Alle leds knipperen
2	Led 1 knippert een keer
3	Led 2 knippert een keer
4	Led 3 knippert een keer
5	Led 4 knippert een keer
6	Led 1 aan
7	Led 2 aan
8	Led 3 aan
9	Led 4 aan
10	Leds gaan roterend oplichten3)
11	Knipper selectie
12	Langzaam knipperen
13	alt

3.3 HTML

Listing 5 laat een deel van de HTML code zien. Alle elementen zijn ondergebracht in een tabel en ieder element heeft een unieke id zodat hiernaar gerefereerd kan worden vanuit het JavaScript om zodoende eigenschappen van het element aan te passen. Bijvoorbeeld het groen worden van een vlakje als een bepaalde knop op een controller is ingedrukt.

```
<!-- Xbox controller 1 table -->
<div class="box">
    <table style="width:100%" id="controller_0">
        <tr>
            <th style="height:50px;text-align:center;background:#00ccff" cla
        </tr>
        <tr>
            <td class="btnStatus" id="AVAIL_0" colspan="4">Available</td>
        </tr>
        <tr>
            <td class="btnStatus" id="D_UP_0">D-up</td>
            <td class="btnStatus" id="D_DN_0">D-dn</td>
            <td class="btnStatus" id="D_LEFT_0">D-left</td>
            <td class="btnStatus" id="D_RIGHT_0">D-right</td>
        </tr>
        <tr>
```

Listing 5: HTML

4 De CGI applicatie

Naam: xbcweb
Builden: $../xbcweb/make clean pi
Installeren: $../xbcweb/make install (Raspberry Pi moet aangesloten zijn en moet het IP adres 10.0.0.42 hebben. Root wachtwoord van de Pi wordt gevraagd).

Applicatie wordt geïnstalleerd in: /var/www/cgi-bin

4.1 Werking

De CGI applicatie is eenvoudig van opzet. Als vanuit de website de CGI applicatie wordt aangeroepen zal deze worden opgestart en de main routine uitvoeren. Nadat de main routine een logfile heeft aangemaakt begint deze direct met het terug geven van een HTML header om vervolgens de parameters uit te lezen die door de website zijn meegegeven. Zie hiervoor listing 7.

```
if ((strCmd = getenv("QUERY_STRING")) != NULL && *strCmd != '\0')
{
    removeSpaces(strCmd);
    retreiveCommand(&cmd, strCmd);
```

Listing 7: Ontvangen en ontleden van parameters

```
typedef struct
{
    uint8_t id;
    char cmd[MAX_CMD_LEN];
    int16_t val;
}command;
```

Listing 6: cmd struct

De parameters worden opgeslagen in de variabele strCmd. strCmd wordt vervolgens ontdaan van eventuele spaties zodat de functie retreiveCommand de string om kan zetten in een cmd struct. Listing 7 laat zien hoe deze struct gedefinieerd is.

Zoals te zien is in listing 8 wordt er in een *if, else if* structuur gekeken welk commando vanuit de website is gegeven. Op basis van dit commando wordt de benodigde actie uitgevoerd.

```
        if (strstr(strCmd, CMD_BTNS))
        {
            if(cmd.id >= 0 && cmd.id < MAX_DEVS)
            {
                startButtonSHM(SHM_NAME);
                readButtonFromSHM(&btn, cmd.id);
                printButton(&btn);
            }
        }
        else if (strstr(strCmd, CMD_LED))
        {
            startQueue();
            sendCommand(&cmd);
            stopQueue();
        }
        else if (strstr(strCmd, CMD_RUMBLE))
        {
            startQueue();
            sendCommand(&cmd);
            stopQueue();
        }
    }
    return (0);
}
```

Listing 8: Afvragen commando's

Als de website de status van de controller buttons opvraagt, CMD_BTNS, dan zal de applicatie het shared memory openen om vervolgens op basis van de index, 0 t/m 3, de data uit dit SHM te lezen. Als de status van een controller is gelezen wordt deze met de functie printButton naar de website gestuurd. Deze functie wordt hierna beschreven. Als er vanuit de website een commando wordt gegeven om een led of de rumble motor aan te sturen, dan de applicatie dit commando in de message queue plaatsen zodat deze door de Deamon kan worden verwerkt. Als een commando is verwerkt zal de applicatie sluiten.

Zoals gezegd stuurt de functie printButton de controller data naar de website. Listing 9 laat een deel van deze functie zien. De functie stuurt dus de controller data gescheiden door komma's als een lange string terug naar de website.

```
void
printButton(button *btn)
{
    printf("%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d",
        btn->avail,
        btn->D_UP,
        btn->D_DN,
        btn->D_LEFT,
        btn->D_RIGHT,
        btn->START,
        btn->BACK,
        btn->LS_PRESS,
        btn->RS_PRESS,
        btn->LB,
        btn->RB,
        btn->LOGO,
        btn->SPARE,
        btn->A,
        btn->B
```

Listing 9: printButton

5 De Deamon applicatie

Naam: xbcdeamon
Builden: $ ../xbcdeamon/make clean pi
Installeren: $ ../xbcdeamon/make install (Raspberry Pi moet aangesloten zijn en moet het IP adres 10.0.0.42 hebben. Root wachtwoord van de Pi wordt gevraagd).

Applicatie wordt geïnstalleerd in: /bin

5.1 Werking

De Deamon applicatie is het meest uitgebreid. De applicatie start met het uitvoeren van de main functie. Vanuit deze functie worden de volgende threads gestart:

- Een device polling thread wordt gestart na aanroep van de functie initDevices;
- Een commandListenerThread wordt gestart vanuit main;
- Een buttonReaderThread wordt gestart vanuit main.

5.1.1 Device polling thread

De device polling thread die gestart is heeft als primaire taak om continue kijken of er nieuwe USB devies worden aangesloten of verwijderd. Zoals in listing 10 te zien is worden vanuit deze thread een aantal functies opgeroepen.

```
void *
pollingThread()
{
    syslog(LOG_INFO, "Device polling thread is started");

    while(running)
    {
        getDevices();
        listClean();
        listUpdateDevices(devices);
        usleep(REFR_TIME_MS * 1000);
    }

    syslog(LOG_INFO, "Device polling thread is stopped");
    return (NULL);
}
```

Listing 10: Pollingthread

Als eerste wordt de functie getDevices aangeroepen. Deze functie heeft als taak om een tijdelijke lijst te creëren met aangesloten USB devices. Vervolgens wordt listClean aangeroepen welke de huishoudelijke taak heeft om devices die niet meer beschikbaar zijn te verwijderen uit de definitieve lijst met devices. Omdat deze definitieve lijst door verschillende threads wordt gebruikt wordt het aanbrengen van wijzigingen thread safe gedaan. Vervolgens wordt de functie listUpdateDevices aangeroepen. Vanuit hier worden devices uit de tijdelijke lijst, onder voorwaarden, en thread safe, toegevoegd aan de definitieve lijst met devices. Op deze wijze is er altijd een definitieve lijst met aangesloten USB devices aanwezig. Om wat rust in de zaak te brengen wordt er als laatste een usleep uitgevoerd. Met de ingestelde REFR_TIME_MS van 100 zal er 100ms gepauzeerd worden.

Tijdens het toevoegen van devices aan de definitieve wordt de functie claimHandle aangeroepen. Zie listing 11. Deze functie zorgt ervoor dat een eventuele door het OS geclaimde handle gedetached wordt om deze vervolgens te claimen voor de applicatie.

```
static uint8_t
claimHandle(libusb_device_handle *handle)
{
    // Find out if kernel driver is attached and if so detach it
    if(libusb_kernel_driver_active(handle, 0) == 1)
    {
        libusb_detach_kernel_driver(handle, 0);
    }

    // Claim interface
    if(libusb_claim_interface(handle, 0) < 0)
    {
        LOGERR("Cannot Claim Interface");
        return (1);
    }
    return (0);
}
```

Listing 11: claimHandle

5.1.2 De commandListenerThread

Listing 12 geeft de code voor deze thread. Zoals te zien is wordt in een *while* loop continue gekeken of er nieuwe commando's binnen komen op de message queue. De commando's worden opgevraagd met de functie getCommand welke daadwerkelijk de message queue uitleest. De ontvangen commando's worden vervolgend in een *if, else if* structuur vergeleken om uiteindelijk de rumble motor of de leds van een controller aan te sturen.

```
void *
commandListenerThread()
{
    command     cmd;

    syslog(LOG_INFO, "Start listening for commands\n");

    /* Start reading the message queue */
    while(running)
    {
        getCommand(&cmd);

        if (strstr(cmd.cmd, CMD_RUMBLE))
        {
            setControllerRumble(getDeviceHandle(cmd.id), cmd.val);
        }
        else if (strstr(cmd.cmd, CMD_LED))
        {
            setControllerLeds(getDeviceHandle(cmd.id), cmd.val);
        }
    }

    syslog(LOG_INFO, "Stopped listening for commands\n");
    return (NULL);
}
```

Listing 12: commandListenerThread

De functie setControllerRumble en setControllerLeds worden respectievelijk weergegeven in listing 13 en listing 14. In beide functies wordt door middel van libusb de controller aangestuurd.

```
int
setControllerRumble(libusb_device_handle *handle, uint16_t speed)
{
    uint8_t     rumble[] = { 0x00, 0x08, 0x00, speed >> 8, speed, 0x00, 0x00, 0x00 };
    int         result, transferred;

    if(handle == NULL)
        return (1); //Return positive since libusb will return negative values

    result = libusb_interrupt_transfer(handle, EP_OUT, rumble, sizeof rumble, &transferred, TIMEOUT);

    return (result);
}
```

Listing 13: setControllerRumble

```
int
setControllerLeds(libusb_device_handle *handle, Leds led)
{
    uint8_t     data[] = { 0x01, 0x03, led };
    int         result, transferred;

    if(handle == NULL)
        return (1); //Return positive since libusb will return negative values

    result = libusb_interrupt_transfer(handle, EP_OUT, data, sizeof data, &transferred, TIMEOUT);

    return (result);
}
```

Listing 14: setControllerLeds

5.1.3 De buttonReaderThread

Listing 15 laat zien dat in de *while* loop voor alle controllers de status van knoppen e.d. wordt opgevraagd. Dit wordt gedaan met de functie getControllerInput welke een struct vult waarin de status van alle buttons en joysticks staat. Als tweede wordt de functie writeButtonToSHM aangeroepen welke de struct naar het shared memory schrijft. Beide functies worden hierna verder beschreven.

```
void *
buttonReaderThread()
{
    buttons btns;
    uint8_t i;

    syslog(LOG_INFO, "Start reading controller buttons\n");

    /* Start reading the buttons of each controller */
    while(running)
    {
        for(i=0; i<MAX_DEVS; i++)
        {
            getControllerInput(getDeviceHandle(i), &btns[i]);
            witeButtonToSHM(&btns[i], i);
        }
    }
    syslog(LOG_INFO, "Stopped reading controller buttons\n");
    return (NULL);
}
```

Listing 15: buttonReaderThread

De getControllerInput functie leest door middel van libusb de gegevens van een controller uit. Listing 16 laat een deel van deze functie zien. In deze functie wordt mede bepaald of een controller beschikbaar is door het struct element avail op 0 of 1 te zetten. Verder worden alle elementen gevuld zodat uiteindelijk de website dit kan visualiseren.

```
int
getControllerInput(libusb_device_handle *handle, button *b)
{
    uint8_t     input[20];
    int         result, transferred;

    if(b == NULL)
        return (1); //Return positive since libusb will return negative values

    if(handle == NULL)
    {
        b->avail = 0;
        return (1);
    }

    b->avail = 1;

    result = libusb_interrupt_transfer(handle, EP_IN, input, sizeof input, &transferred, TIMEOUT);

    b->D_UP     = input[2] & 0x01;
    b->D_DN     = input[2] & 0x02;
    b->D_LEFT   = input[2] & 0x04;
    b->D_RIGHT  = input[2] & 0x08;
    b->START    = input[2] & 0x10;
    b->BACK     = input[2] & 0x20;
    b->LS_PRESS = input[2] & 0x40;
    b->RS_PRESS = input[2] & 0x80;

    b->LB       = input[3] & 0x01;
```

Listing 16: getControllerInput

De functie writeButtonToSHM heeft de verantwoordelijkheid om de controller data in het shared memory te schrijven. Listing 17 geeft deze functie weer. Het schrijven in het shared memory wordt door middel van een semafoor beschermt. Dit is nodig omdat de CGI applicatie dit shared memory uitleest. Op deze wijze wordt dus gegarandeerd dat het lezen en schrijven in het shared memory nooit gelijktijdig kan gebeuren.

```
void
witeButtonToSHM(button *b, uint8_t index)
{
    if(index < 0 || index > MAX_DEVS)
    {
        return;
    }

    /* Begin critical section */
    sem_wait(sem);

    map[index] = *b;

    sem_post(sem);
    /* End critical section */
}
```

Listing 17: writeButtonToSHM

Het shared memory zelf wordt gecreëerd in de functie initButtonSHM welke in de main van de Deamon wordt aangeroepen. Listing 18 laat het deel van de functie zien waarbij het shared memory in het geheugen wordt gemapped. De mapping die wordt gedaan is op basis van een button struct waardoor er eenvoudig data kan worden gelezen of geschreven zoals te zien is in listing 16 hierboven.

```
/* Map shared memory into address space */
if((map = (button *) mmap(NULL, size, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0)) == MAP_FAILED)
{
    LOGERR("mmap() failed");
    return;
}
```

Listing 18: Mappen van het shared memory

6 Script

Naam: S65XbcDaemon
Builden: Niet van toepassing. (Script is wel executeerbaar gemaakt)
Installatie: Kopieer de file op de Raspberry Pi in: /etc/init.d
(Bijvoorbeeld: $ scp S65XbcDaemon root@10.0.0.42: /etc/init.d)

Het script uit listing 19 dient ervoor om de Deamon applicatie te starten, te stoppen of te herstarten. Het script wordt geplaatst in de map zoals hierboven bij installatie is opgegeven.

```
case "$1" in
  start)
    echo "Starting the xbc daemon application."
    /bin/xbcdaemon &
    ;;
  stop)
    echo -n "Stopping the xbc daemon application."
    /usr/bin/killall xbcdaemon
    ;;
  restart|reload)
    "$0" stop
    "$0" start
    ;;
  *)
    echo "Usage: $0 {start|stop|restart}"
    exit 1
esac

exit $?
```

Listing 19: Script voor starten van de Deamon

7 Make files

Iedere applicatie heeft een eigen makefile waarmee de betreffende applicatie kan worden gebouwd. In deze makefile zijn de volgende opties voorzien:

clean:	wist alle object en executable.
all:	build een executable op basis van de gcc compiler
pi:	build een executable op basis van de arm-linux-gcc compiler
info:	geeft een overzicht van de variabelen waarmee make werkt.
Debug:	build een executable op basis van de gcc compiler met daarin debug informatie
install:	Kopieert de betreffende executable naar de Raspberry Pi
backup:	Maakt een archief aan met project bestanden in een daarvoor bestemde map. De naam voor dit archief is als volgt: xbcweb_<datum>_<tijd>.tar. Op deze wijze kunnen dus eenvoudig snapshots gemaakt worden.
html:	produceert html documentatie met Doxygen.

De makefile is vrij generiek van opzet waardoor tijdens de ontwikkeling van de applicaties hierin niets meer hoeft te worden gewijzigd. In listing 20 is te zien welke parameters, ook wel macro's genoemd, nodig zijn om het geheel te laten werken.

```
#
#   Generic Makefile for simple projects.
#
#   (C) 2015, A.W. Janisse
#
#   Macro's:
#       OUTPUT  : Name of the executable
#       PIDIR   : Place for installation on the Raspberry PI
#       CC      : Default compiler. (Note make pi will build for the Raspberry Pi platform)
#       LIBS    : Libraries to use when building
#       CFLAGS  : Compiler flags
#       ZIPDIR  : Directory to put the backup archive files
#

OUTPUT  = xbcdaemon
SUBDIRS = ../common
PIDIR   = root@10.0.0.42:/bin
CC      = gcc
LIBS    = -lusb-1.0 -lrt -pthread
CFLAGS  = -O2 -Wall -Werror
ZIPDIR  = ../backup

### -----[ Do not change anything below this line ]----- ###
```

Listing 20: Project specifiek instellingen makefile

Voor de precieze werking kan het best worden gekeken in de makefile zelf. Alles is hierin voldoende gedocumenteerd om de werking te kunnen begrijpen.