FACULTY OF SCIENCES

Standardizing USB in the WebAssembly System Interface

Wouter Hennen
Student number: 01905957

Supervisors: Prof. dr. Bruno Volckaert, Prof. dr. ir. Filip De Turck
Counsellors: Dr. ing. Merlijn Sebrechts, ing. Michiel Van Kenhove, Dr. ing. Tom Goethals

Master's dissertation submitted in order to obtain the academic degree of
Master of Science in Computer Science

Academic year 2023-2024

GHENT UNIVERSITY

Acknowledgements

I would like to thank my counsellors Dr. ing. Merlijn Sebrechts and ing. Michiel Van Kenhove for their help, commitment, feedback and time to make this thesis possible.

I would also like to thank my supervisors, Prof. dr. Bruno Volckaert and Prof. dr. ir. Filip De Turck.

Additionally, I would also like to thank my collegue Warre Dujardin for the collaboration on the proposal.

I would like to thank the WASI community for the creation of WIT, the available tooling and support.

Disclaimer regarding the master's thesis

This master's thesis is part of an examination. Any comments made by the evaluation committee during the oral presentation of the master's thesis were not incorporated into this text.

Abstract

This thesis explores how support for USB devices can be added to the WebAssembly System Interface using the Component Model. This allows developers to communicate with USB devices in a platform-independent and language-independent manner, similar to the current capabilities of WebUSB & WebAssembly on browsers. The built-in access control in WASI ensures that this can be done securely, allowing users to control access to USB devices.

Extended Abstract

Your Name
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

Counsellor's name
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

Advisor's name
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

***Abstract*—This document is a placeholder for your extended abstract. Create a new Overleaf project to write your extended abstract, download the PDF from that project, and include it in your master's thesis.**

***Index Terms*—template**

I. How to add the extended abstract

You should write the extended abstract as a separate overleaf project. Then compile it there, download the PDF, and upload it to this project.

Use the "IEEE conference proceedings template" to create the extended abstract project.

Then download the final PDF, upload it to the root of the Overleaf project of your master's thesis, open the file `chapters/4_extended_abstract.tex`, and point the statement in that file to the PDF you just uploaded.

Contents

List of Figures

List of Tables

List of Acronyms

API Application Programming Interface.

CPU Central Processing Unit.

IOT Internet of Things.

K8S Kubernetes.

USB Universal Serial Bus.

WASI WebAssembly System Interface. , 1, 2

Wasm WebAssembly. , 1

WIT Wasm Interface Type. , 1

1

Introduction

Updating software of IOT devices is currently difficult. As the devices are home appliances, they are often used for longer periods of time, compared to other tech such as smartphones. As a consequence, these devices need longer software support. In reality, a lot of devices stop getting updates after a few years, making them vulnerable to cyberattacks. When an IOT device gets hacked, it can be used as part of a botnet, or worse, send sensitive info (such as a video feed) to the attacker.

There are multiple causes to why this software support is short: no standardized updating mechanisms, special hardware and compilers, etc.

In Webbrowsers, WebAssembly (Wasm) acts as a generalized layer between the native browser API and a programming language. This model can also be applied outside the browser context and can solve some of the current problems of current native code.

However, in order to do so, some changes are needed. As Wasm is primarily targeted towards browser applications, it only provides libraries to do those things, for example accessing the DOM. No low-level libraries exist for controlling hardware devices, which would be required for IOT devices. To solve this, a new specification was created: the WebAssembly System Interface (WASI)

WASI is a group of API specifications that can be used with software compiled to Wasm. These APIs provide a standardized interface for any programming language that can compile to Wasm. Because the APIs act as a layer between the native APIs and the application, access control can be applied to control what an application can access. To provide APIs that feel natural in any language, a new interface description language was born: Wasm Interface Type (WIT). For each programming language, bindings can be created that expose a generalized API.

With WASI, standardized APIs can be created that can be used to talk and control hardware. As the APIs are not device specific anymore and no compiling to specific architectures is needed anymore, updating software for IOT devices has become a lot easier.

1 Introduction

Problem

The WASI standard is still in development and is not finished yet. One of the APIs that is currently missing is one for USB devices. Having an API for USB devices is crucial for IOT devices, as a lot of hardware, such as cameras, will oftentimes communicate over USB.

Goal

The goal of this master's thesis is to research the possibilities to add a new USB API to WASI. Afterwards, a new proposal will be created to add the new API to the official standard. Special attention will be given to access control, so that programs don't have access to all devices by default, reducing a possible attack surface. Benchmarks will test the performance of a WASI USB implementation compared to native code and Wasm in the browser.

Research Questions

2

Background

In the early days of computers, various connectors existed for computers to communicate with external devices. A few examples of such connectors are the PS/2 port (Primarily used for Mouse / Keyboard) or the Parallel port (Often used for printers). These connectors have varying sizes, shapes and limitations and cannot be used for all devices. Therefore, computers needed to have all these ports, requiring lots of space. To address these issues, the USB connector was introduced. YEars later, the interface has become the de-facto standard for wired device communication. Over the years, USB has evolved with multiple revisions. These revisions focus on key aspects of the interface, such as transfer speeds, power delivery and added functionality. The USB connector has also undergone revisions, making it smaller and reversible, so it is usable on a larger variety of devices.

In this thesis, the focus is on the software side of the USB standard. Therefore, the hardware will not be considered further.

2.1 Transferring Data

2.1.1 Pipes

Data is transferred through pipes. A pipe is a connection from the host controller to the endpoint. Not all pipes are the same: they differ in the bandwidth they support, which transfer types are supported, in which direction data can flow, and their packet and buffer size. Pipes can generally be split up into two kinds.

Streaming Pipes

A Streaming Pipe is a one-way communication channel for the host or guest device to send any kind of data to the other end. This pipe is controlled by either the host of or guest device, and data is sent in a sequential way. The isochronous, interrupt and bulk transfer types will use this pipe to send data.

Message Pipes

A Message Pipe is a bidirectional communication channel. This pipe allows both the host and guest device to send commands in either direction on the same pipe. All message pipes are controlled by the host device. Only one transfer type supports this pipe: the control transfer type.

2.1.2 Transfer Types

The USB standard defines four transfer types. Each transfer type serves a different purpose, being optimized for speed, latency, correctness or reliability.

Interrupt Transfer

Interrupt transfers are most used for devices that transfer small amounts of data frequently. They have a bounded latency and are therefore suited for devices that require low latency and low bandwidth. Interrupt transfers can be initiated by both the host and guest device. The sent data will be queued by the sender, until the receiver polls the device.

Examples of devices that often use interrupt transfers are mice and keyboards.

Isochronous Transfer

Isochronous transfers are used for real-time data streaming. They provide a guaranteed data rate, but do not guarantee a correct transfer of data, and data can be lost. This makes the transfer type not suitable for situations where data integrity is important.

Examples of use cases where isochronous data transfer is often used is for audio devices or video streaming.

Bulk Transfer

Bulk transfers are suited for transferring large amounts of data where timing is not an issue, but data integrity is. No guarantee on timing is made, but guaranteed correct delivery is.

Examples of use cases for bulk transfers are sending files to printers or storage devices.

Control Transfer

Control transfers are used for configuration, command and status operations between the host and guest device. Control transfers operate on the Message pipe and has setup, data and handshake stages. The handshakes guarantee correct delivery, but will lead to a slower transfer speed. Therefore, control transfers are not used to transfer a lot of data, but rather for device initialization and control.

The Control transfer type can be seen as a TCP connection but for USB devices.

2.1.3 Descriptors

2.2 LibUSB

2.3 WebUSB

The WebUSB specification allows webpages to control non-standard USB devices. Browsers already provide easy APIs for common USB devices, such as mice, keyboards, cameras and microphones. However, accessing devices that do not follow these common USB use cases were not supported in the browser. In 201x, the WebUSB specification was created to provide a new API that allows this.

WebUSB has been proven useful in a lot of cases. For example, it has been used to control arduinos and upload new programs to these devices. Another kind of use case is to easily upgrade devices through USB, without the requirement to install special software.

For example, after discontinuing its game streaming service Stadia, Google provided a firmware update for its Stadia controllers to enable bluetooth support. This update could be installed by connecting the controller to the computer and following the steps on the website. No additional software needed to be installed, making it very user-friendly.

WebUSB is not part of the web standards and is currently only supported in Chromium-based browsers. WebUSB only provides a limited javascript API and is an abstraction over the raw USB interface, making it less useful for IOT devices.

2.4 Docker

2.5 WASI

2.6 WIT

3

Architecture

In dit hoofdstuk ...

3.1 Sectie titel

Vul aan...

4

Implementation

In dit hoofdstuk ...

4.1 Sectie titel

Vul aan ...

4.2 Sectie titel2

Vul aan ...

4.2.1 Subtitel

Vul aan ...

5

Evaluation

In dit hoofdstuk ...

5.1 Sectie titel

Vul aan ...

5.2 Sectie titel2

Vul aan ...

5.2.1 Subtitel

Vul aan ...

6

Conclusie

7

Future Work

In dit hoofdstuk ...

7.1 Sectie titel

Vul aan ...

7.2 Sectie titel2

Vul aan ...

7.2.1 Subtitel

Vul aan ...

Referenties

Bijlagen

Bijlage A

Toelichting bijlage.

Bijlage B

Toelichting bijlage.