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.

Wasm WebAssembly. , 1

List of Acronyms

1

Introduction

In today's world, hyperscale cloud providers, such as Microsoft Azure and Amazon Web Services, are used extensively. These companies provide serverless functions, which run for short bursts and close immediately afterwards. In order to do this efficiently, the sandbox in which the code is executed should be able to quickly spin up, so the cold start problem can be minimized. Existing services like Docker and K8S are mostly used for this, but are not ideal: while being more performant than full blown operating systems, there cold boot is still problematic. A new standard, the WebAssembly System Interface (WASI), has been created which should greatly mitigate this problem.

WebAssembly (Wasm) originally started as a way for browsers to execute code written in a language other than Javascript. It allows code from any supported programming language to be compiled to machine instructions for a virtual CPU, and for these instructions to be efficiently executed in the browser sandbox. This has proven to be successful and has opened numerous possibilities for web apps that weren't possible before. A great example of this is the new Google Earth website, which uses WebAssembly underneath, which allows for performant rendering of the Earth.

This model of running code on a lightweight virtual CPU seen interest outside of the browser, mainly for servers, low power devices (IOT), and so on. There, the WebAssembly System Interface could replace Docker, and provide performance improvements, power and storage savings.

WASI is currently still in development, and has a long way to go to get stable. One of the problems WASI is currently facing is the lack of APIs available to interface with the system. One of the APIs that is currently missing is the one to interact with USB devices. Having such an API would be useful for servers, cars, IOT devices, etc. The goal of this master's thesis is to create an USB proposal for expanding the WASI, and eventually add support for USB to WASI. With access control, users can decide which devices a program running in WASM can access.

1.1 Objectives

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

3

Titel derde hoofdstuk

In dit hoofdstuk ...

3.1 Sectie titel

Vul aan...

4

Titel vierde hoofdstuk

In dit hoofdstuk ...

4.1 Sectie titel

Vul aan ...

4.2 Sectie titel2

Vul aan ...

4.2.1 Subtitel

Vul aan ...

Conclusie

Vul aan…

Ethische en maatschappelijke reflectie

Deze sectie is enkel vereist voor de opleidingen industrieel ingenieur. De locatie van deze sectie lichtjes af van de volgorde voorgeschreven door de faculteit. Wij raden aan om deze reflectie als deel van de conclusie te maken omdat je daardoor eenvoudig kan refereren naar resultaten in je masterproef zelf.

Meer informatie kan je opzoeken op https://www.sdgs.be/nl/sdgs

Referenties

Bijlagen

Bijlage A

Toelichting bijlage.

Bijlage B

Toelichting bijlage.