HEAVY METAL ON BARE METAL

* 

= 

Portieren für Einsteiger

Ahmad Fatoum – a.fatoum@pengutronix.de

Über mich

Ahmad Fatoum

Pengutronix e.K.

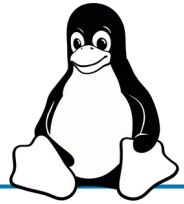
a3f

a.fatoum@pengutronix.de

- Kernel-/Bootloader-Portierung
- Treiberentwicklung
- Systemintegration
- Beratung rund um Linux auf eingebetteten Systemen

Eingebettete Systeme

Warum Embedded Linux?

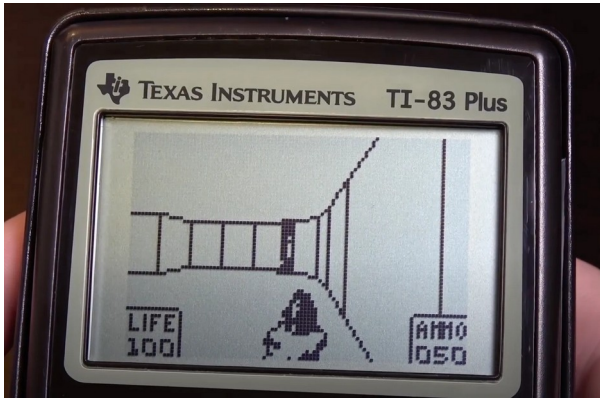


- Linux als *die* HW-Abstraktionsschicht
- Keine Lizenzgebühren
- Zugriff auf Quelltext
- Embedded Linux *ist* Linux → viel Software
  - Browser
  - GUI Toolkits
  - Laufzeitumgebungen für Programmiersprachen

  → Vieles aus Server- und Desktop-Bereich wiederverwendbar

DOOM

- 1993: erstmals für MS-DOS erschienen
- 1997: Linux-Port Quelltext veröffentlicht
- 1999: GNU-GPL-Reliziensierung des Quelltexts
- Seit dem viele viele verschiedene Ports und Klone
  - https://itrunsdoom.tumblr.com



chocolate-doom: native

- (SDL-basierertes) chocolate-doom ist guter Startpunkt

```
$ git clone https://github.com/chocolate-doom/chocolate-doom
```

- Buildsystem erkennt Eigenheiten der Plattform

```
$ autoreconf -fi && ./configure && make
```

- Am Ende hat man eine ausführbare Datei

```
$ src/chocolate-doom -iwad DOOM1.WAD
```

chocolate-doom: cross compiling

- Programm lässt sich auch vorab auf dem Entwicklungsrechner kompilieren

```
$ ./configure                                  \
    --build=x86_64-linux-gnu                   \
    --host=arm-linux-gnueabihf
```

- --build: Wo läuft der Compiler?
- --host: Wo soll die kompilierte Anwendung laufen?

Abhängigkeiten

- chocolate-doom enthält nur die Spiellogik
  - Der Rest ist in externe Bibliotheken ausgelagert
- Mit den GNU binutils kann man schauen was dem Linker fehlte:

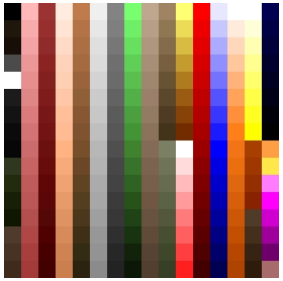
```
$ objdump -p src/chocolate-doom | grep NEEDED
  NEEDED               libSDL2-2.0.so.0
  NEEDED               libSDL2_mixer-2.0.so.0
  NEEDED               libSDL2_net-2.0.so.0
  NEEDED               libc.so.6
```

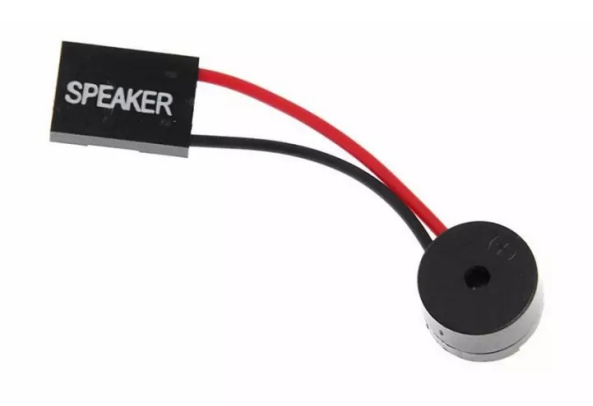
- Und welche Symbole diese exportieren

```
$ nm -D src/chocolate-doom
         U fread
         U SDL_RenderCopy
         U malloc
… u.v.m
```

Abhängigkeiten: SDL

- Gute Video Alternative zu direktem x11 im Original-DOOM
- Unterstützt verschiedene Betriebssysteme
  - Linux, Windows, AmigaOS, macOS, FreeBSD, Haiku, ... etc.
- chocolate-doom benutzt es für
  - Videoausgabe (DOOM rendert in Software; Optimierung: Farbpaletten)
  - Audioausgabe (MIDI; alternativ: PC-Speaker)
  - Spielereingabe (Keyboard, Joystick, Mouse)
  - Timer einstellen (Aufwachen zum nächsten Frame)
  - Netzwerk für Multiplayer-Modus (UDP)





Abhängigkeiten: C Standard Bibliothek

- Der C Standard unterscheidet zwischen Hosted Implementation:
  - Zeichenkettenmanipulation (`memcpy`, `tolower`, ...)
  - Speicherverwaltung (`malloc`, `free`, ...)
  - Dateimanipulation (`fopen`, `fread`, `printf`, ...)
  - Programmkontrolle (`getenv`, `exit`, ...)
  - Und mehr, umso mehr unter UNIX: GNU libc hat >1500 Symbole
- Und Freestanding Implementation:
  - Dateitypen und Größen
  - Paar Makros für Compiler `__builtin`s
  - Der Rest ist Dein Problem
- C-Bibliotheken wie newlib erleichtern es Hosting nachzurüsten

```
void _exit();
char **environ;
caddr_t sbrk(int incr);
int open(const char *name, int flags, ...);
int fstat(int file, struct stat *st);
int stat(const char *file, struct stat *st);
int lseek(int file, int ptr, int dir);
int read(int file, char *ptr, int len);
int gettimeofday(struct timeval *p,
                 struct timezone *z);
```

Abhängigkeiten: Fließkommaarithmetik

C

- In Skripting-Sprachen Status Quo, aber:
- Fließkomma-Operationen...
  - ...können langsam und platzaufwendig sein,
    wenn die Hardware sie nicht unterstützt
  - ...bedürfen manchmal nicht-portable initiale Konfiguration
  - ...lassen sich für DOOM mit Festkommazahlen ersetzen
- Festkommazahlen...
  - ...sind normale Zahlen → nativ vom Prozessor unterstützt
  - ...sorgen in unserem Fall für weniger Portierungsaufwand

Abhängigkeiten: Loader

- Gegenstück zum Linker
- Was passiert vor `main()`?
  - Runtime Linker (`ld.so`) und Entry Point (`crt0.o`) teilen sich die Arbeit:
    - Verteilung des Programms an die richtigen Stellen im Hauptspeicher
    - Nullen von Globalen Variablen
    - Mit externen Symbolen verlinken
    - Stack aufsetzen für lokale Variablen
    - Konstruktoren ausführen, z.B. um C-Bibliothek zu initialisieren
    - `main()` mit Argumenten aufrufen und auffangen
  - Auf Bare Metal
    - Normalerweise kein dynamisches Linken → Programm enthält alles was es braucht
    - Startup Code setzt alles nötige auf

Source-Port Mindestanforderungen

- C Compiler und Laufzeit-Umgebung
- Framebuffer
- Eingangssignale
- Pulsgenerator (Optionale Soundeffekte)
- Teile der Standard C-Bibliothek

Source-Port Mindestanforderungen

- C Compiler und Laufzeit-Umgebung → in C geschrieben
- Framebuffer → Bootsplash
- Eingangssignale → Menü/Knöpfe
- Pulsgenerator (Optionale Soundeffekte) → Piepen/Helligkeit
- Teile der Standard C-Bibliothek → Verfügbar

→ Erkenntnis: Der Bootloader kann schon das nötige!

bareDOOM: Anatomie eines Portes

- C Compiler
  - barebox v2021.08.0 unterstützt 10 Befehlssätze

- arm32	- x86_64	- powerpc	- Kalray MPPA
- thumb2	- riscv32	- mips	- sandbox
- arm64	- riscv64	- openrisc	

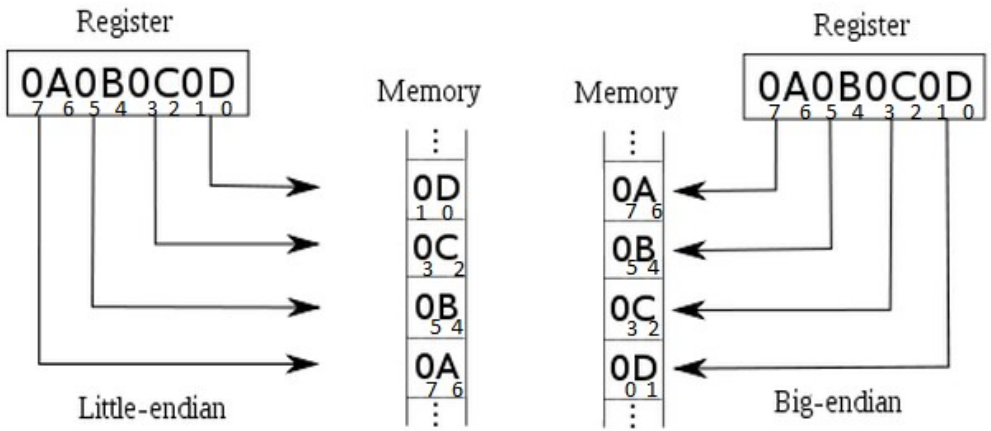
  - Plattformauswahl analog zum Linux-Kernel:

```
ARCH=riscv CROSS_COMPILE=riscv64-linux-gnu- make my_config
```

    - `CROSS_COMPILE` ist Präfix vom Cross-Compiler, falls vorhanden
    - `arch/$ARCH/Makefile` kümmert sich um die Compileroptionen
    - `my_config` enthält Basiskonfiguration für Plattform

bareDOOM: Anatomie eines Portes

- C Compiler: Fallstricke
  - Little- vs Big-Endian
  - 64-bit Inkompatibilitäten
  - Floating Point



- → In vielen Ports schon gelöst, z.B. doomgeneric

doomgeneric

- Wie newlib aber für DOOM. Man implementiere:
  - `DG_Init` → Initialisierung, z.B. von Video Hardware
  - DG_DrawFrame → DG_ScreenBuffer ausgeben
  - DG_SleepMs → Anzahl Millisekunden schlafen
  - DG_GetTicksMs → Zeit in Millisekunden abfragen
  - DG_GetKey → Neue Tastatureingabe abfragen
  - Hosted C Implementierung für den Rest
- Und schon hat man einen funktionierenden Source-Port

bareDOOM: Anatomie eines Portes

- Laufzeit-Umgebung
  - Eigener Startup-Code:
    - Globale Variaben nullen
    - Code Relocation
    - Stack aufsetzen
    - Großteils in C geschrieben (GNU macht`s möglich)
  - objcopy(1) erstellt flaches Programmabbild mit Befehlen ab offset 0

bareDOOM: Anatomie eines Portes

- Framebuffer:
  - Grafiken könnte einfach nach `/dev/fb*` ge-blit-tet werden

    z.B. `cp /dev/hwrng0 /dev/fb0`
  - Helper für Farbtransformation
  - Asynchron zum Vertikalen Refresh

    → Tearing ϟ

bareDOOM: Anatomie eines Portes

- Eingangssignale

  barebox Input-Framework:

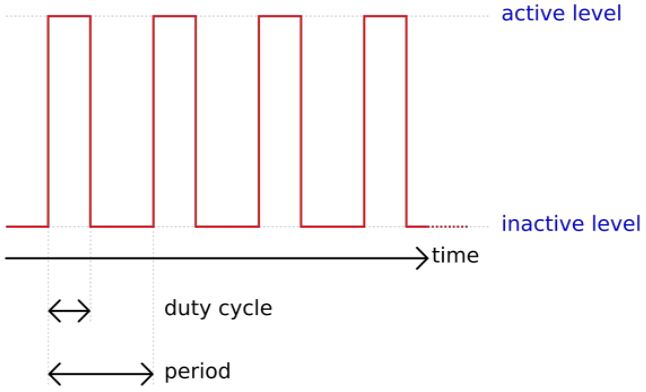
  - Macht Debouncing und Key-Mapping
  - Hardware: USB keyboard, GPIO, Virt I/O...
  - Abstraktion für Anwendungen:

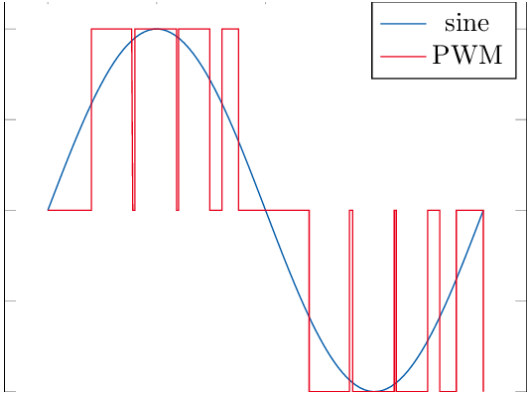
    Key Up und Down Events

```
gpio_keys {
    compatible = "gpio-keys";
    #address-cells = <1>;
    #size-cells = <0>;

    button@0 {
        linux,code = <KEY_LEFT_CTRL>;
        gpios = <&gpio 1 GPIO_ACTIVE_HIGH>;
    };

    button@1 {
        linux,code = <KEY_SPACE>;
        gpios = <&gpio 2 GPIO_ACTIVE_HIGH>;
    };
};
```

bareDOOM: Anatomie eines Portes

- Pulsgenerator (PWM)
  - barebox unterstützt PWM, v.a., für Display-Helligkeit
  - Neues Sound-Framework um GRUB-Piepsequenzen im Hintergrund abzuspielen.
  - DOOM IWAD enthält separate Pieptonspur mit 140 samples die Sekunde → Quasi das gleiche





bareDOOM: Anatomie eines Portes

- Teile der Standard C-Bibliothek
  - Viele barebox-Treiber stammen aus dem Linux Kernel
    - Teile der Kernel Hilfsfunktionen im barebox verfügbar
    - Keine normgerechte C Bibliothek, aber etwas Überlappung
  - Einige barebox-Kommandos stammen aus Linux Userspace
    - Teile der ISO/POSIX Standard-Bibliothek vorhanden
    - Was für DOOM fehlte
      - Buffered File I/O → Implementierung ohne Puffer
      - exit → entferne green Thread aus der Warteschlange

Und nun?

- Fazit: DOOM läuft nun auf barebox

  Und wie bekommst du DOOM auf dein Gerät?

Und nun?

- Fazit: DOOM läuft nun auf barebox

  Und wie bekommst du es auf dein Gerät?

  → Natürlich indem du barebox selbst portierst ;)

barebox portieren

- Ausführbare Firmware bauen
- Hardware initialisieren
- Rest des Bootloaders nachladen
- (Video-)Treiber implementieren

- Man nehme mal den Nintendo Gamebov Advance als Beispiel

barebox portieren

- Ausführbare Firmware Bauen
  - barebox muss für ARMv4T Befehlssatz gebaut werden
  - GBA BIOS führt ARM-Instruktion in ersten 4 byte aus
  - Danach muss das Nintendo Logo im Programmcode eingebettet sein
  - Danach kommt etwas Metainformation und dann der Code,
    der von den ersten 4 byte angesprungen wird

Der Prebootloader (PBL)

- Das Ideal: barebox startet komplett im RAM, Hardware-Beschreibung kommt von außen
  - Unter Linux teils schon Realität:

    Multi-Platform Kernel-Image und Device-Tree, ermöglicht durch Bootloader
  - Was tun wenn man selbst Bootloader ist? Prebootloader schreiben!
- PBL initialisiert das nötigste an Hardware (Caches, Interrupts, Clocks, RAM controller,...)
  - auf dem GBA macht das ROM BIOS was gemacht werden muss

Der Prebootloader (PBL)

- PBL lädt den Rest des Bootloaders nach
  - GamePAKs sind read-only. Ungünstig wenn man Globale Variablen haben will

    → Prebootloader muss Rest vom Bootloader von GamePAK nach RAM laden
  - Multiboot-Kabel: Code läuft direkt aus dem kleinen 32K RAM

    → Prebootloader muss komprimierten barebox in den großen (256K) RAM laden
  - Gleiches Ergebnis: barebox „proper“ läuft immer komplett im RAM
    - RAM zu klein? Dann braucht man auch keinen Bootloader,

      weil Linux eh nicht reinpassen wird…
- PBL reicht den korrekten Device Tree weiter

  → barebox wiederverwendbar für verschiedene Hardware-Varianten

barebox Treiber

- Hört sich komplexer an als es sein kann. Beispiel GBA Video Treiber:

```
#include <common.h>
#include <fb.h>

#define GBA_VRAM_BASE  ((void *)0x6000000)
#define GBA_MMIO_BASE  IOMEM(0x4000000)

#define DISPCNT  0
#define   DISPCNT_BGMODE(x)   ((x) & 0x3)
#define   DISPCNT_DISPENABLE(x)  (((1 << 8) << x) & 0x1F00)

static int gba_lcdfb_probe(struct device_d *dev)
{
    static struct fb_ops gba_lcdfb_ops;
    static struct fb_info fbi = {
        .fbops = &gba_lcdfb_ops,
        .screen_base = GBA_VRAM_BASE,
        .xres = 240, .yres = 160, .bits_per_pixel = 16,
        .blue.length = 5, .green.length = 5, .red.length = 5,
        .blue.offset = 0, .green.offset = 5, .red.offset = 10,
    };
    writel(DISPCNT_DISPENABLE(2) | DISPCNT_BGMODE(3), GBA_MMIO_BASE + DISPCNT);
    return register_framebuffer(&fbi);
}

static struct driver_d gba_lcdfb_driver = {
    .name  = "gba_lcdfb",
    .probe = gba_lcdfb_probe,
};
device_platform_driver(gba_lcdfb_driver);
```

Neugier geweckt?

- Webseite: https://barebox.org
- Entwicklung findet per Mailing-Liste statt

  https://lists.infradead.org/mailman/listinfo/barebox
- `#barebox` auf Libera.Chat IRC mit Matrix-Bridge

  https://app.element.io/#/room/#barebox:matrix.org
- Probiere die Browser-Demo aus:

  https://barebox.org/jsbarebox/
- Oder lokal unter Linux mit `ARCH=sandbox`:

```
user@host$ git clone https://git.pengutronix.de/git/barebox
user@host$ cd barebox
user@host$ make sandbox_defconfig
user@host$ make
user@host$ ./barebox
```

bare/box|doom/ auf neuer Hardware?

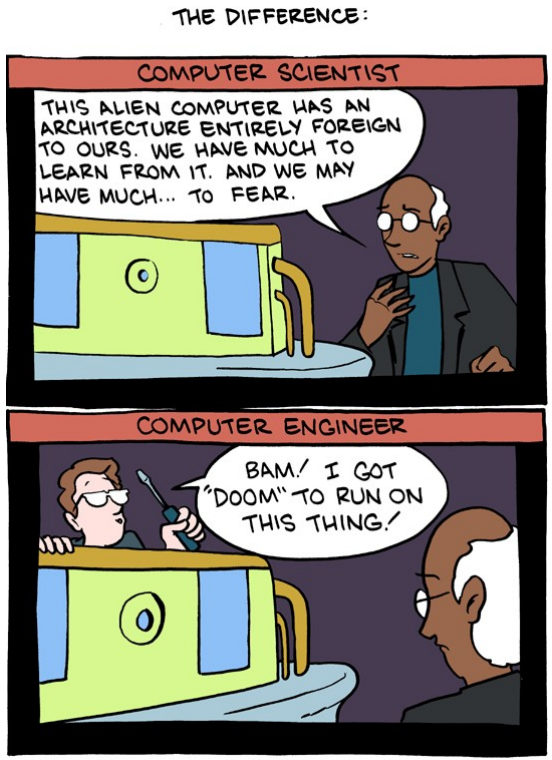
- Ja, bitte!
  - Wie bootet barebox einen modernen ARM-Prozessor?

    → From Reset Vector to Kernel
  - Wie konfiguriere und benutze ich barebox für ein neues Board?

    → Beyond “just” Booting: barebox Bells and Whistles
  - The barebox Porter’s Guide
- Community:
  - Schildere Dein Problem im IRC oder auf der Mailing Liste

    und es findet sich sicher was ähnliches zum Abschreiben
  - Und wenn es dann mal funktioniert:
    - Gerne Neue Plattformen upstreamen:

      Läuft ein Betriebssystem drauf? Braucht es einen Bootloader? Dann schick die Patches!
    - Für jeden barebox Port gibt es einen DOOM-Port geschenkt



bareDOOM: Wie geht's weiter?

- Ursprünglich als Aprilscherz gedacht
- Freue mich trotzdem sehr über Patches

  https://github.com/a3f/bareDOOM
  - MIDI Musik?
  - barebox sound im Web browser?
  - Bessere Performance?
  - GBADoom statt doomgeneric?
  - Was euch sonst so einfällt!

Bildquellen

- DOOM-Logo: https://snipstock.com/image/doom-png-images-17-87390
- Digital Kamera: https://www.backscatter.com/Canon-6D-Camera
- E-Book Reader: https://de.aliexpress.com/item/32810693705.html
- Gateway: http://www.artila.com/en/p_matrix.html
- Traktor: https://www.deere.de/de/agrar-management-systeml%C3%B6sungen/empf%C3%A4nger-und-displays/gen-4-integrated-displays/
- Waschmaschine: https://danube-deutschland.de/professionelle-ausstattung/
- DOOM auf Kamera: https://knowyourmeme.com/memes/it-runs-doom
- DOOM auf Taschenrechner: https://www.inputmag.com/tech/doom-the-best-worst-places-it-has-run/amp
- DOOM auf Oszi: https://www.youtube.com/watch?v=OU16lIx_pC8
- SDL: https://commons.wikimedia.org/wiki/File:Simple_DirectMedia_Layer,_Logo.svg
- PC-Speaker: https://www.lazada.com.ph/products/pc-motherboard-internal-speaker-i877726238.html
- Sine/PWM: https://www.researchgate.net/figure/PWM-signal-with-switching-frequency-of-f-s-500-Hz-generating-a-sine-wave-of-50-Hz_fig1_323796384
- Admiral Ackbar: https://redpepper.land/blog/its-a-trap/
- Endianness: https://en.wikipedia.org/wiki/File:Big-little_endian.png
- Nintendo GBA: https://commons.wikimedia.org/wiki/File:Nintendo-Game-Boy-Advance-Purple-FL.jpg
- Comic: https://www.smbc-comics.com/index.php?db=comics&id=2158#comic

Danke für die Aufmerksamkeit!

Fragen?