A

BOOTBOOT

Protokoll

Specifikáció és Kézikönyv

Első kiadás

2017 - 2020

Szerzői jog

Tartalom

Előszó

“A kezdet kezdetén kell a leggondosabban ügyelni rá, hogy meglegyen a dolgok egyensúlya.”

/ Frank Herbert /

Az elmúlt évtizedekben nagyot változott a személyi számítógépek világa, és ahogyan a gépek indulnak. A 64 bit megjelenésével, először a történelem során, a memóriatartomány meghaladta a tárolókapacitását. Ez alapvető változásokat hozott a firmvereknél is.

A tárolókapacitás szintén nőtt, ha nem is Moore törvénye alapján, de meredeken ívelve. A régi partícionálási módok elavultak, új táblákat alakítottak ki, melyek közül egy lett a de facto sztenderd.

Sajnálatos módon az a firmver, ami az új partíciós táblát hozta, annyira túlbonyolított és túlkomplikált, hogy sok gyártó még csak meg sem próbálta beépíteni a gépeibe (különösen a kis erőforrású gépek esetén). Ennek következtében mégsincs de facto sztenderd indulás, különböző hardverek különböző, inkompatíbilis módon indulnak. Nem mindenki vette át az új táblát sem. Hogy még rosszabb legyen a helyzet, sok rendszernek meg kellett őriznie a visszamenőleges kompatíbilitást ősrégi gépekkel is.

Vannak próbálkozások az operációs rendszer indítás egységesítésre, de sajnos olyan összetett és túlbonyolított módon, hogy azt önmagában is operációs rendszernek nevezhetnénk.

Ezért alkottam meg ezt az egységes operációs rendszer indítást leíró specifikációt, és írtam meg a referencia implementációkat több különböző platformra. A cél az, hogy mire ezek a kicsi platformfüggő programok lefutnak, egy egységes, platformfüggetlen, 64 bites környezet álljon rendelkezésre, mely képes ugyanabból a C forrásból, ugyanazzal a linkelő szkripttel fordított programot futtatni. A forrás és az előre lefordított binárisok (beleértve a minta C kernelt) letölthetők innen:

`https://gitlab.com/bztsrc/bootboot`

Ezek a referencia implementációk Nyílt Forráskódúak és mindenféle garancia nélküli Szabad Szoftverek, abban a reményben íródtak, hogy hasznosnak bizonyulnak.

Baldaszti Zoltán Tamás

Szándékosan üres oldal

Bevezetés

Amikor egy gépet bekapcsolunk, először az operációs rendszert kell betölteni. Vannak szofisztikált programok, melyek lehetővé teszik a több rendszer kiválasztását is ugyanazon a gépen, mint például a GRUB. Ezeket rendszerindító menedzsereknek hívják. A BOOTBOOT nem ilyen. Ez egy rendszerbetöltő, aminek az a célja, hogy egységes 64 bites környezetet biztosítson különböző platformokon (a "BOOTBOOT" bájtsorozat tárolásához 64 bit szükséges). Ha több rendszer közül is szeretnél választani egy gépen, akkor egy BOOTBOOT kompatíbilis opciót kell beállítanod valamelyik rendszerindító menedzserben, hogy BOOTBOOT kompatíbilis operációs rendszert indíthass. Ha megfelel, hogy egy gépen egy rendszer van csak, akkor nincs szükség rendszerindító menedzserre, a rendszerbetöltő önmagában elég.

Egy operációs rendszer sokféleképp betölthető. Lehet ROM-ban, flashen, lemezen, SD kártyán, soros vonalról, hálózatról stb. A BOOTBOOT Protokoll nem definiálja ezt. Azt sem köti ki, milyen formátumban kell lennie a memórialemeznek. Ezek időről időre és rendszerről rendszerre változnak.

A protokoll megköveteli ugyanakkor, ha az operációs rendszer lemezen van tárolva, akkor a GUID Partíciós Táblát kell használnia (vagy bármelyik későbbi sztenderd táblát). Mivel nem minden firmver képes ezt kezelni, ezért a betöltő kötelessége ezt elrejteni és az operációs rendszert megtalálni a lemezen. Így a felhasználóknak nem kell az eltérő firmverekker bajlódniuk, ha olyan rendszerlemezről akarnak indítani, ami egy másik gépen lett partícionálva és formázva.

Néhány szó az operációs rendszer kernel formátumáról. Jelen sorok írásakor nincs de facto sztenderd, de van két általánosan elterjedt formátum: az Executable and Linkable Format, és a Portable Executable formátum. Nem lenne ildomos azt állítani, hogy az egyik jobb, mint a másik, végtére is mindkettő ugyanazokat az adatokat tartalmazza, csak máshogy. Ezért mindkettőt támogatja a protokoll. Ha valamelyik (vagy egy leendő harmadik) szabványossá válna, akkor a protokollt felül kell bírálni, hogy csak egy lehetőséget biztosítson, így a felhasználóknak nem kellene törődnie a formátum kompatíbilitással sem.

Végezetül jelen dokumentum felosztásáról pár szó. Két részre lett osztva: az első fele általánosságban tárgyalja a protokollt, míg a második fele a referencia implementációkat taglalja részleteikben.

Szándékosan üres oldal

Specifikáció

A dokumentum ezen fele a BOOTBOOT Protokollról szól.

Az operációs rendszer betöltése

A rendszer betöltés kifejezés sok mindent takarhat, de lényegében arról szól, hogy átadódjon a vezérlés - a környezeti információkkal egyetemben - az operációs rendszer kernelére.

Az operációs rendszerek típusairól

Két általánosan elterjedt kernel típus van. Az első mindent egyben tartalmaz, többnyire statikusan linkelt kép formájában (monolitikus kernel). A második több darabra van szétvágva, ezáltal a prvilegizált kód egy kicsi kernelben (mikrokernel, exokernel, hypervisor stb.) található, míg minden más ki lett tolva felhasználói szintre, ami általában külön fájlt is jelent (de nem szükségszerűen, lásd Minix).

Mindkettőnek lehet induló memórialemeze. Ebben találhatók azok a fájlok, melyekre az induláskor már szükség van. Monolitikus típusnál ez a kép a kernellel együtt töltődik be, és (mivel a meghajtók a kernelben vannak), opcionális. Másrészről a mikrokernelek esetén egy ilyen memórialemezkép nélkülözhetetlen, mivel minden feladat kódja szeparált fájlokban található.

A BOOTBOOT Protokoll alkotója az OS fejlesztők jelentős többségével egyetemben azon az álláspontón van, hogy a mikrokernel architektúra sokkal biztonságosabb és rugalmasabb, ezért a BOOTBOOT Protokoll elsősorban mikrokernelekre lett tervezve.

Az induló memórialemezkép

Mivel a protokoll a mikrokernelekre fókuszál, melyeknek szükségük van több fájlra, ezért az induló memórialemez (initrd) kötelező. És mivel a lemezkép mindenképp betöltésre kerül, ezért a kernelt is benne helyezi el. Ez nem általánosan elterjedt gyakorlat, de egyetlen fájlra redukálja a szükséges fájlok számát, akárcsak egy monolitikus rendszer esetén. Megjegyzendő, hogy a protokoll eléggé rugalmas ahhoz, hogy egy statikusan linkelt kernelt (mint például a Minix) töltsön be “lemezképként”.

A lemezkép tömöríthető. A referencia implementációk a gzip deflate tömörítést támogatják, de más is lehet, amennyiben a tömörítés mágikus bájtokkal beazonosítható. Egy BOOTBOOT kompatíbilis betöltő transzparensen kicsomagolja a lemezképet a memóriába. Mivel az egész lemezkép betöltésre kerül, érdemes a méretét alacsonyan (pár megabájton) tartani.

A kicsomagolt memórialemez formátuma nem része a protokollnak. Minden egyes operációs rendszer azt használhat, ami a legjobban megfelel a céljainak. Emiatt a BOOTBOOT Protokoll csak egy API-t vár el, ami képes kezelni a lemezképet, és ennek hiányában egy B-tervet biztosít.

A boot partíció

A protokoll nem definiálja, hogy hol kell lennie az induló lemezképnek, csupán annyit vár el, hogy egy BOOTBOOT Protokoll kompatíbilis betöltő el tudja érni, és ki tudja csomagolni a memóriába. A referencia implementációk soros vonalat, ROM-ot és indító (boot) partíciót is képesek kezelni.

A protokoll elvárja, hogy lemez esetén a partíciós tábla GUID Partíciós Tábla legyen. Ennek oka az interoperabilitás biztosítása a kölönböző operációs rendszerek között.

A boot partíció egy kicsi partíció a lemez elején. Tartalmazhat a firmver számára fontos fájlokat is, de ami a protokoll számára lényeges, hogy itt található az induló memórialemezkép.

Ha a boot partíció rendelkezik fájlrendszerrel, akkor a kompatíbilitás miatt FAT16 vagy FAT32 formázottnak kell lennie. Sok firmver (mint például az UEFI és a Raspberry Pié) szintén elvárja ezt. Ha a boot partíció firmver fájlokat is tartalmaz, akkor "EFI System Partition" típusúnak, vagy röviden ESP-nek kell lennie. Ez azért van, mert a GPT az EFI firmver (most már UEFI) részeként lett definiálva. Ebben a felállásban a memórialemezkép a következő fájl a boot partíción:

BOOTBOOT\INITRD

vagy több architektúra támogatás esetén (élő operációs rendszer lemezképeknél):

BOOTBOOT\(arch)

mint pl. **BOOTBOOT\X86_64** vagy **BOOTBOOT\AARCH64**.

Ha a firmver partíció nem használ fájlrendszert, nem kezeli a FAT16/32 fájlrendszereket, illetve speciális típust vár el (ezért az ESP nem használható), akkor a firmver partíció és a boot partíció két külön, elszeparált partíció lesz.

Ez esetben az operációs rendszer fejlesztőjének két lehetősége van: vagy csinál egy másik partíciót FAT fájlrendszerrel, BOOTBOOT könyvtárral, benne a memórialemezképpel; vagy a memórialemezképet direktben a partícióra is rakhatja. Bármelyiket is választja, a boot partíciót *EFI_PART_USED_BY_OS*-nek (2-es bit a GPT Partíció Bejegyzés attribútum jelzőjében) kell jelölni.

Nem szabad elfelejteni, hogy a memórialemezkép teljes egészében betöltődik, ezért ha a teljes boot partíciót elfoglalja, akkor annak a partíciónak kicsinek (pár megabájtosnak) kell lennie.

Ha a lemezképet soros vonalon akarod betölteni, akkor kelleni fog Goswin von Brederlow *raspbootcom*-ja (https://github.com/mrvn/raspbootin), vagy az USBImager (https://gitlab.com/bztsrc/usbimager) ami egy Windows-on, MacOSX-en és Linux-on is elérhető grafikus alkalmazás.

Fájlrendszer meghajtók

Amint korábban említettük, a BOOTBOOT Protokoll nem írja elő a memórialemez formátumát, helyette fájlrendszer meghajtókat használ egyetlen API hívással:

```
typedef struct {
    uint8_t *ptr;
    uint64_t size;
} file_t;

file_t myfs_initrd(uint8_t *initrd, char *filename);
```

A referencia implementációk forrásában ezek elkülönített, **fs.h** (vagy **fs.inc**) nevű fájlban találhatók. Minden memórialemezkép formátumhoz pontosan egy függvény tartozik.

Minden függvény megkapja az induló memórialemezkép címét, valamint egy nullával lezárt, ASCII fájlnév címét. Ha ilyen nevű fájlt megtalálható, akkor a függvény visszaad egy struktúrát a fájl tartalmának a címével és méretével. Ha szükséges, a függvény ehhez foglalhat le memóriát. Hiba esetén (ha a lemezkép formátumát nem ismeri fel, vagy a fájl nem található) akkor {NULL, 0}-t kell visszaadnia. A BOOTBOOT Protokoll elvárja, hogy egy kompatíbilis betöltő sorra meghívja a meghajtók függvényeit, míg valamelyik érvényes eredménnyel nem tér vissza.

Ha minden meghajtó függvény {NULL,0}-t adott vissza, akkor egy B-terv lép életbe. Ez megkeresi a legelső, adott architektúrán futtatható programot a memórialemezképben. A fájl jogosultságok nem számítanak ez esetben, csak a futtatható fájl fejléce. Ez a B-terv lehetővé teszi, hogy ismeretlen formátumú memórialemezképeket is kezeljen a betöltő, vagy hogy statikusan linkelt kernelt töltsön be.

Ha valamelyik fájlrendszer meghajtó támogatja a memórialemezkép formtumát, akkor a kernel neve megadható a *kernel* kulcsú környezeti változóban.

A referencia implementációk a következő archívum és fájlrendszer formátumokat támogatják:

- *statikusan linkelt futtatható* (minden fájl egyetlen futtathatóba szerkesztve)
- *ustar*
- *cpio* (hpodc, newc és crc variáns)
- *FS/Z* (az OS/Z natív fájlrendszere)
- *SFS* (osdev.org bizonyos tagjai által kitalált fájlrendszer)
- *James Molloy's initrd* (valamiért népszerű a hobbi OS fejlesztők körében)

Bármelyik másik fájlrendszer támogatása bármikor hozzáadható, egy kivétellel. A FAT fájlrendszer nem használható memórialemezképek esetén. Ez nem komoly megszorítás, mivel úgysem lenne hatékony, az ustar vagy cpio sokkal jobb választás erre a célra.

Kernel formátuma

A kernel futtatható formátuma vagy **Executable and Linkable Format** (ELF), vagy **Portable Executable** (PE). Mindkét esetben a formátumnak 64 bitesnek kell lennie (*ELFCLASS64* az ELF-nél és *PE_OPT_MAGIC_PE32PLUS* a PE-nél).

A kombinált kód és adatszegmens natív 64 bites kódot kell tartalmazzon, és negatív címre kell szerkeszteni. A referencia implementációk esetén ez *EM_X86_64 (62)* vagy *EM_AARCH64 (183)* lehet ELF-ben, és *IMAGE_FILE_MACHINE_AMD64 (0x8664)* vagy *IMAGE_FILE_MACHINE_ARM64 (0xAA64)* ha PE. Az **x86_64** architektúrát a BIOS / Multiboot / El Torito és UEFI betöltők használják, míg az **AArch64** a Raspberry Pi 3 és 4 esetén támogatott.

Protokoll szintek

Hogy hova lesz leképezve a kernel, az a protokoll szintjétől függ. A renferencia implementációk az 1. szintet, a *PROTOCOL_STATIC*-ot valósítják meg. A 2. szint, a *PROTOCOL_DYNAMIC* még a jövő zenéje. A 0. szint, a *PROTOCOL_MINIMAL* beágyazott rendszereknél használatos, ahol a környezet és minden cím bedrótozott, és a frémbuffer lehet nincs is támogatva a hardverben.

Statikus

Azok a betöltők, amik az 1. szintet tudják, a kernelt és a többi komponenst a linkelő szkripttel összhangban fix címre képezik le (lásd “Gépállapot” fejezet). Ebben a specifikációban a továbbiakban az egyszerűség kedvéért a statikus protokollt használjuk. Előrekompatíbilitás miatt, minden BOOTBOOT kompatíbilis kernelnek tartalmaznia kell a 2. szint által elvárt szimbólumokat.

Dinamikus

A második szintű betöltők ugyanakkor a memóriatérkép címeit a kernel szimbólumtáblájából olvassák ki. Csak abban különbözik az 1. szinttől, hogy a címek flexibilisek (de továbbra is a negatív tartományra korlátozódnak, és laphatárra kell esniük):

- A kernel a futtható fejlécében szereplő *Elf64_Ehdr.p_vaddr* vagy *pe_hdr.code_base* címre kerül.
- A bootboot struct a *bootboot* szimbólum címére lesz leképezve.
- A környezeti sztring az *environment* szimbólum címére lesz leképezve.
- A lineáris frémbuffer az *fb* szimbólum címére lesz leképezve.
- Végezetül az MMIO terület az *mmio* szimbólum címére lesz leképezve.

Belépési pont

Amikor a BOOTBOOT kompatíbilis betöltő végzett, a vezérlést az *Elf64_Ehdr.e_entry* illetve a *pe_hdr.entry_point* mezőben szereplő címre adja át.

Környezeti változók

Ha a boot partíció FAT fájlrendszerű, akkor a környezeti változókat tartlamazó fájl neve

BOOTBOOT\CONFIG

míg ha a memórialemez képe elfoglalja a teljes partíciót, akkor a fájlrendszer meghajtókat használva

sys/config

lesz. Amennyiben ez utóbbi nem megfelelő az operációs rendszer számára, a név átírható a bootboot forrásában. A környezeti változók mérete limitálva van egy lapkeretben (4096 bájt).

A konfiguráció a kernelnek újsor (‘\n’ vagy 0xA) szeparált, “kulcs=érték” párokat tartalmazó, nullával lezárt UTF-8 sztringként kerül átadásra. C típusú kommentek megengedettek. A BOOTBOOT Protokoll csupán két kulcsot definiál, a *screen* és *kernel* kulcsokat, a többit szabadon választhatja az operációs rendszer kernele (és eszközmeghajtói). Példa:

```
// BOOTBOOT Opciók (environment)

/* --- Betöltő specifikus --- */
// kívánt felbontás, ha nincs megadva, autodetektált
screen=800x600
// elf vagy pe bináris a memórialemezképben
kernel=sys/core

/* --- Kernel specifikus, amit csak szeretnél --- */
amitakarsz=valami
masikcucc=enabled
valamiszam=100
valamicim=0xA0000
```

A *screen* paraméter a kívánt képernyőfelbontás. Ha nincs megadva, akkor a képernyő natív felbontása alapból, vagy 1024x768 ha ez utóbbi nem ismert. A legkissebb érvényes értéke a 640x480.

A *kernel* paraméter alapból sys/core és a kernel futtatható neve a memórialemezben. Ha ez nem megfelelő, akkor beállítható, vagy az alapértelmezett a bootboot forrásában módosítható.

Az átmeneti változók a környezet végére kerülnek (az UEFI parancssorból). Ha ugyanaz a kulcs többször is előfordul, akkor mindig a legutolsó előfordulás a mérvadó.

Indulási problémák esetén a környezet módosításához egy másik gépbe kell átrakni a lemezt, vagy egy egyszerű OS-t kell indítani, mint a DOS, és módosítani a **BOOTBOOT\CONFIG**-ot a boot partíción egy egyszerű szövegszerkesztővel. UEFI esetén az EFI Shell *edit* parancsa is használható, vagy a

parancssorban lehet "kulcs=érték" párokkal felülbírálni a beállításokat (a parancssori kulcsok elsőbbséget élveznek a fájlbeliekkel szemben).

A környezeti változók sztringje a kernel elé kerül leképezésre a memóriában, a linker által megadott címre. A kernelben a következő definícióval lesz elérhető:

```
extern unsigned char *environment;
```

A `bootboot` struct struktúra

A bootboot struct definíciója a bootboot.h fájlban található, ami innen tölthető le:

`https://gitlab.com/bztsrc/bootboot/blob/master/bootboot.h`

A legfontosabb struktúra, amiben a betöltő adatokat ad át a kernel-nek. Define őrrel és extern “C” kulcsszóval van ellátva, hogy C++-ban írt kernelek is használhassák.

A struktúra maga egy 128 bájtos fejlécből, és egy változó hosszúságú memóriatérképből áll, amiben minden bejegyzés 16 bájtos. A fejléc első 64 bájtja minden architektúrán azonos, míg a második 64 bájt platform specifikus, rendszerleíró táblák címeit tartalmazza.

Fejléc mezők

Platform független

```
uint8_t    magic[4];     // 0x00-0x03
```

A mágikus *BOOTBOOT_MAGIC* bájtsorozat, “BOOT”.

```
uint32_t   size;         // 0x04-0x07
```

A bootboot struct teljes mérete. Legalább 128 bájt, tartalmazza a memória térképet.

```
uint8_t    protocol;     // 0x08
```

Ez a mező az alsó 2 biten a BOOTBOOT Protokoll szintjét tartalmazza (amit a struktúrát összeállító betöltő használt). Vagy *PROTOCOL_STATIC (1)* vagy *PROTOCOL_DYNAMIC (2)*. Ha a 7. bit (előjelbit) 1 azaz *PROTOCOL_BIGENDIAN (0x80)*, akkor a struktúra nagy-elöl (big-endian) számokat tartalmaz. A 2 – 6 bitek tárolják a betöltő típusát, *LOADER_BIOS (0)*, *LOADER_UEFI (1)* vagy *LOADER_RPI (2)* jelenleg. Jelen specifikáció jövőbeli verziói új értékeket definiálhatnak.

```
uint8_t    fb_type;      // 0x09
```

A frémbuffer formátuma, *FB_ARGB (0)* -tól *FB_BGRA (3)* -ig. A leggyakoribb az FB_ARGB, ahol a legalacsonyabb helyiérték a kék, a legmagasabb pedig nem használt (lfb-n nincs alfa csatorna) kicsi-elöl (little-endian) formátumban.

```
    uint16_t   numcores;    // 0x0A-0x0B
```

A CPU magok száma. SMP rendszereken nagyobb, mint 1.

```
    uint16_t   bspid;       // 0x0C-0x0D
```

Az indító processzor azonosítója (BootStrap Processor ID, a Local APIC ID x86_64 esetén).

```
    int16_t    timezone;    // 0x0E-0x0F
```

A detektált időzóna, ha az támogatott a platformon, percekben *-1440* és *1440* közötti szám. Nem befolyásolja a *datetime* mező értékét (ami mindig egységes greenwichi középidőben van, UTC-ben).

```
    uint8_t    datetime[8]; // 0x10-0x17
```

A rendszerindítás időpontja UTC-ben, binárisan kódolt decimális formátumban, ha a platformon van valós idejű óra (RTC). Az első két bájt az év, 0x20 0x17, egy bájt a hónap 0x12, egy bájt a nap 0x31. Ezt követi az óra 0x23, perc 0x59 és másodperc 0x59 bájt. Az utolsó bájtban 1/100-ad másodperc tárolható, de támogatás hiányában a legtöbb platformon 0x00. A *timezone* mezőtől független érték.

```
    uint64_t   initrd_ptr;  // 0x18-0x1F
    uint64_t   initrd_size; // 0x20-0x27
```

A memórialemezkép fizikai címe és mérete a memóriában (mindig a pozitív címtartományban).

```
    uint8_t   *fb_ptr;      // 0x28-0x2F
    uint32_t   fb_size;     // 0x30-0x33
```

A frémbuffer fizikai címe és mérete bájtokban. Nem azonos a linker által definiált *fb* virtuális címmel.

```
    uint32_t   fb_width;    // 0x34-0x37
    uint32_t   fb_height;   // 0x38-0x3B
    uint32_t   fb_scanline; // 0x3C-0x3F
```

A frémbuffer felbontása, soronkénti bájtszáma a memóriában (részletekért lásd a “Lineáris frémbuffer” fejezetet).

Platformfüggő mutatók

A fejléc második 64 bájtja mindig architektúrafüggő. Ezek a következők x86_64 esetén:

```
uint64_t x86_64.acpi_ptr;    // 0x40-0x7F
uint64_t x86_64.smbi_ptr;
uint64_t x86_64.efi_ptr;
uint64_t x86_64.mp_ptr;
uint64_t x86_64.unused0;
uint64_t x86_64.unused1;
uint64_t x86_64.unused2;
uint64_t x86_64.unused3;
```

AArch64 esetén pedig a következők. Az *mmio_ptr* a BCM2837 MMIO fizikai címe (a kernel számára az *mmio* virtuális címre van leképezve):

```
uint64_t aarch64.acpi_ptr;   // 0x40-0x7F
uint64_t aarch64.mmio_ptr;
uint64_t aarch64.efi_ptr;
uint64_t aarch64.unused0;
uint64_t aarch64.unused1;
uint64_t aarch64.unused2;
uint64_t aarch64.unused3;
uint64_t aarch64.unused4;
```

Memóriatérkép bejegyzések

```
MMapEnt    mmap;             // 0x80-0xFFF
```

Platformfüggetlen memóriatérkép. Ha szükséges, a bejegyzések száma a következőképp kiszámítható:

num_mmap_entries = (bootboot.size – 128) / 16;

A bejegyzésekben tárolt információk pedig a következő C makrókkal nyerhetők ki:

MMapEnt_Ptr(a) = a memóriaterület címe
MMapEnt_Size(a) = a memóriaterület mérete bájtokban
MMapEnt_Type(a) = a memóriaterület típusa 0 – 15
MMapEnt_IsFree(a) = igaz, ha a memóriaterület az operációs rendszer számára felhasználható.

A típus lehet *MMAP_USED (0)*, *MMAP_FREE (1)*, *MMAP_ACPI (2*), *MMAP_MMIO(3,* memóriába leképzett B/K*)*. Bármilyen más értéket *MMAP_USED*-ként kell kezelni.

A bootboot struct a kernel elé kerül leképezésre a memóriában, a linker által megadott címre. A kernelben a következő definícióval lesz elérhető:

```
extern BOOTBOOT bootboot;
```

Lineáris frémbuffer

A frémbuffer formátuma mindig 32 bites tömörített színkód pixelenként, lehetőség szerint ARGB (kék a legkissebb helyiértéken). A felbontás natív lesz, vagy 1024x768, ha ez nem ismert. A kívánt felbontást a környezeti változókban, az environment sztringben a *screen=SZÉLESSÉGxMAGASSÁG* formában lehet megadni. Ha az ARGB mód nem támogatott, akkor a csatornák sorrendjét a *bootboot.fb_type* tartalmazza.

A frémbuffer a többi MMIO területtel együtt a kernel elé kerül leképezésre a memóriában, a linker által megadott címre. A kernelben a következő definícióval lesz elérhető:

```
extern uint8_t fb;
uint32_t *pixel = (uint32_t*)(&fb + offset);
```

Az (X, Y) képernyőkoordináták a következőképp számíthatók át eltolás értékké:

*offset = (bootboot.fb_height – Y) * bootboot.fb_scanline + 4 * X*.

Habár a *bootboot.fb_size* mező 32 bit, az 1. szintű betöltők statikus *fb* címe valahol 4096 x 4096 pixelben limitálja a felbontást (a képaránytól és a soronkénti bájtszámtól is függ). Ez több, mint elég az Ultra HD 4K (3840 x 2160) felbontáshoz. A 2. szintű betöltők dunamikusan a kernel *fb* szimbólum címére képezik le a frémbuffert, ezért mentesek ettől a korlátozástól.

Gépállapot

Amikor a kernel átveszi a vezérlést, a soros vonali debug konzol fel van konfigurálva, hardver megszakítások letilva és a kód rendszerfelügyeleti szinten fut. A lebegőpontos (FPU) és SIMD utasítások engedélyezve vannak bizonyos szintig (SSEx, Neon). SMP szintén, minden mag fut (lásd Függelék). A virtuális leképezés az MMU-ban bekapcsolva, és a memóriatérkép a következő:

A RAM (16G-ig) identikusan van leképezve a pozitív tarományban (felhasználói vagy alsó fél memória, USER SPACE). A negatív címek a kernel tarmtománya, és nem privilegizált módból nem elérhető (felső fél, KERNEL SPACE).

A kitömörített **induló memórialemezkép** teljes egészében az identikusan leképzett tartományban van, és a bootboot struct *initrd_ptr* valamint *initrd_size* mezői jelölik a helyét.

A képernyő 32 bites pixelformátumú, **lineáris frémbuffer**e a negatív címtérben, az *fb* szimbólumnál a **-64M** vagy **0xFFFFFFFF_FC000000** címen kezdődik (a többi **MMIO** pedig **-128M** vagy **0xFFFFFFFF_F8000000** címnél, ha támogatott, a fizikai címe az *mmio_ptr* mezőben). A frémbuffer fizikai címét a *fb_ptr* mező tárolja.

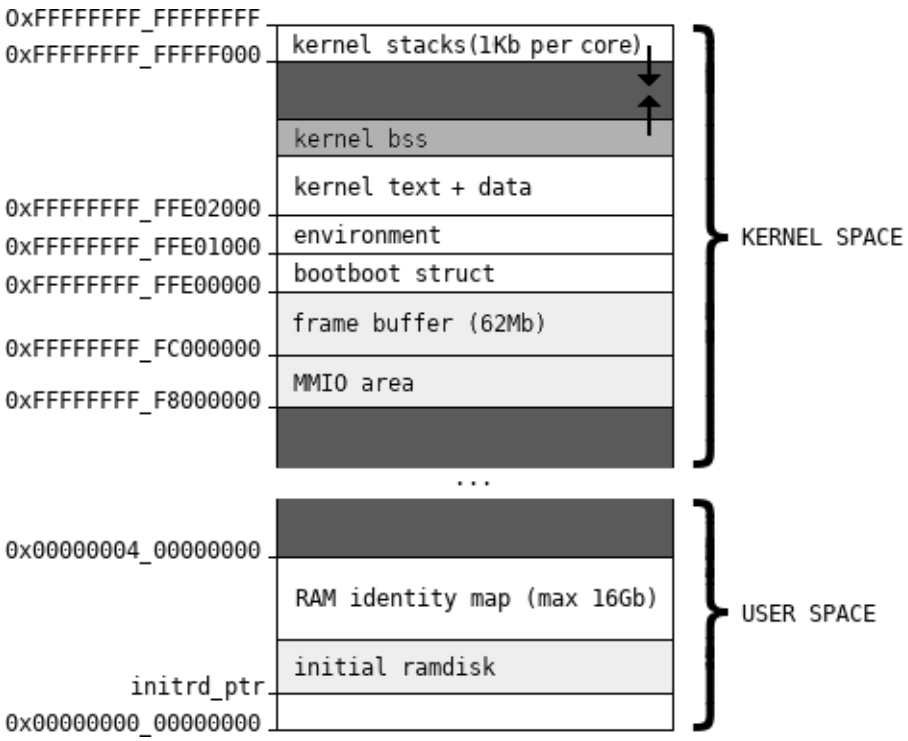
A fő információs **bootboot struct** a *bootboot* szimbólumnál, **-2M** vagy **0xFFFFFFFF_FFE00000** címen van leképezve.

A környezeti változókat tartalmazó sztring az **environment** szimbólumnál, **-2M + 1 lap** vagy **0xFFFFFFFF_FFE01000** címen elérhető.

A kernel kombinált **kód és adat szegmense** a **-2M + 2 lap** vagy **0xFFFFFFFF_FFE02000** címre kerül leképezésre. Ez után a szegmens után, linker által meghatározott címen található a **bss adat szegmens**, melyet a betöltő kinulláz. Az 1. szintű betöltők a kernel méretét 2M-ben limitálják, beleértve az infókat, kód, adat, bss és verem szegmenseket. Ennek több, mint elégnek kell lennie bármelyik mikrokernel számára. Ha a kernel külön csak olvasható kódszegmenst és nem futtatható adatszegmens szeretne a biztonság növelése érdekében, akkor felülbírálhatja a lapfordító táblákat, amint a vezérlést megkapta. A BOOTBOOT Protokoll csak egy betölthető szegmenst kezel az egyszerűség és a hordozhatóság kedvéért (amit *boot*-nak hív a példa linker szkript, lásd Függelék).

A **kernel verem** a memória tetején található, **nullá**nál indul és lefele nő. Az **első pár lap**ot a betöltő képezi le, a többit a kernelnek kell, amint szüksége lesz rá. Minden CPU magnak külön 1k-s induló verme van SMP rendszereken.

A linker szkript által definiált címek használata elég egyszerű (nincs szükség API-ra és az ABI sem számít), ezért platformfüggetlen módon tudja általa a betöltő átadni az információkat a kernelnek.



***Ábra**: memóriatérkép vezérlés átadáskor, nem méretarányos. A sötétszürke régiók nincsenek leképezve*

Referencia implementációk

A dokumentáció második fele a referencia implementációkról szól. Ez kézikönyvként is szolgál, és a felhasznált firmver függvénykönyvtár hivatkozásokat is tartalmazza.

Mindegyik implementáció szabadon letölthető a következő címről:

`https://gitlab.com/bztsrc/bootboot`

- **x86_64-bios**: IBM PC BIOS / Multiboot / El Torito / Linux boot implementáció
- **x86_64-uefi**: IBM PC UEFI implementáció
- **aarch64-rpi**: Raspberry Pi 3 / 4 implementáció
- **mykernel**: egy minta BOOTBOOT kompatíbilis kernel teszteléshez

***Ábra**: A minta kernel képernyője referenciának*

IBM PC BIOS / Multiboot / El Torito / Linux boot

BIOS (http://www.scs.stanford.edu/05au-cs240c/lab/specsbbs101.pdf) alapú rendszereken ugyanaz a kép betölthető MBR-ből (GPT hibrid indítás), lánctöltéssel VBR-ből, futtatható ROM-ból; vagy Multiboot (https://www.gnu.org/software/grub/manual/multiboot/multiboot.html) illetve Linux boot használatával (https://elixir.bootlin.com/linux/latest/source/Documentation/x86/boot.txt).

Induló memórialemez

Lehet BIOS Bővítő ROM-ban (max ~96k). Nem valami sok, de lehet tömörített is. Lemezről a BIOS INT 13h / AH=42h funkcióival tölti be.

Memóriatérkép

A memóriatérképet a BIOS INT 15h / AX=0E820h funkció hívásával kéri le.

Lineáris frémbuffer

A frémbuffer beállításához a VESA 2.0 VBE, INT 10h / AH=4Fh funkcióit használja.

Gépállapot

A20 kapu engedélyezve, soros konzol COM1-en INT 14h / AX=0401h-val inicializálva 115200,8N1-re. Indítási idő lekérése INT 1Ah. IRQk letiltva. GDT nem meghatározott, de érvényes, IDT nincs beállítva. SSE és SMP inicializálva. A kód felügyeleti szinten, **0-ás gyűrű**n fut minden magon.

Korlátozások

- Mivel maga a betöltő védett módban fut, ezért csak az első 4G RAM-ot tudja leképezni.
- A CMOS nvram nem tárol időzónát, ezért mindig GMT+0 a *bootboot.timezone*.
- Nem támogatja az AES-256-CBC enkriptált initrd-ket, csak az SHA-XOR-CBC-t.

Indítás

- **BIOS lemez / cdrom**: másold le a *bootboot.bin*-t ***FS0:\BOOTBOOT\LOADER*** néven. Partíción kívülre is helyezheted (`dd conv=notrunc seek=x`). Telepítsd a *boot.bin*-t az El Torito CDROM boot katalógusába, a Master Boot Record-ba (vagy Volume Boot Record-ba ha van boot menedzsered), és mentsd le a bootboot.bin első szektorának címét 0x1B0 dwordre. A *mkboot* ezt csinálja (https://gitlab.com/bztsrc/bootboot/blob/master/x86_64-bios/mkboot.c).
- **BIOS ROM**: telepítsd a *bootboot.bin*-t ***BIOS Bővítő ROM***-ba.
- **GRUB**: add meg a *bootboot.bin*-t Multiboot "kernel"-nek a grub.cfg-ban, vagy lánctöltheted a *boot.bin*-t. Mind az initrd mind az environment modulként betölthető (lásd Függelék).

IBM PC UEFI

UEFI gépeken (http://www.uefi.org/) az operációs rendszert egy szabványos EFI OS betöltő alkalmazás tölti be.

Induló memórialemez

Lehet ROM-ban (max 16M) mint PCI Option ROM. Az EFI_PCI_OPTION_ROM_TABLE protokoll használatával és direkt eléréssel, mágikus bájt próbával detektálja.

Lemezről az EFI_SIMPLE_FILE_SYSTEM_PROTOCOL-al ha fájl a memórialemezkép, illetve BLOCK_IO_PROTOCOL-al ha GPT partíció. Ez az implementáció támogatja mind az SHA-XOR_CBC és AES-256-CBC enkriptált initrd-ket egyaránt.

Memóriatérkép

A memóriatérképet az EFI_GET_MEMORY_MAP boot idejű szolgáltatás adja.

Lineáris frémbuffer

A frémbuffer beállításához az EFI_GRAPHICS_OUTPUT_PROTOCOL-t (röviden GOP-ot) használja.

Gépállapot

Konzolhoz SIMPLE_TEXT_OUTPUT_INTERFACE-t használ, ami soros vonalra is küldhető. Indítási idő lekérése EFI_GET_TIME. IRQk letiltva. GDT nem meghatározott, de érvényes, IDT nincs beállítva. SSE és SMP inicializálva. A kód felügyeleti szinten, **0-ás gyűrű**n fut minden magon.

Korlátozások

- A betöltő és initrd PCI Option ROM-okat digitálisan alá kell írni ahhoz, hogy működjenek.

Indítás

- **UEFI lemez**: másold le a *bootboot.efi*-t ***FS0:\EFI\BOOT\BOOTX64.EFI*** néven.
- **UEFI ROM**: használd a *bootboot.rom*-ot, ami a ***PCI Option ROM*** képe a bootboot.efi-nek.
- **GRUB, UEFI Boot Menedzser**: add hozzá a *bootboot.efi*-t az opciókhoz.

Raspberry Pi 3 / 4

Raspberry Pi (https://www.raspberrypi.org/) alaplapokon az SD kártya boot partíciójáról a bootboot.img-t kernel8.img néven a start.elf tölti be.

Induló memórialemez

Nincs ROM támogatás, de az initrd soros vonalról is betölthető. A lemezképet egy, a bootbootban implementált EMMC SDHC meghajtó tölti be. Gzip nem ajánlott, mert lassabb, mint a kártyaolvasás.

Memóriatérkép

A memóriatérképet kézzel állítja össze, a VideoCore MailBox tulajdonságok csatornájáról gyűjtött adatok alapján. A szokásoson túl a **BCM2837 MMIO** is le van képezve a kernel memóriába **-128M** vagy **0xFFFFFFFF_F8000000** címen. Fizikai címe a *bootboot.aarch64.mmio_ptr* mezőben található, ami a MIDR_EL1 rendszer regiszter alapján kerül megállapításra.

Lineáris frémbuffer

A frémbuffer VideoCore MailBox üzenetekkel kerül beállításra.

Gépállapot

Soros vonali debug konzol az UART0-án (PL011), 115200,8N1-re állítva és az USB debug kábel a GPIO 14 / 15-ös lábára csatlakoztatva. Lebegőpontos utasítások engedélyezve. A kód felügyeleti szinten, **EL1-**n fut mind a 4 magon.

Korlátozások

- 1G RAM-ot képez le (Raspberry Pi 3-on nincs is több, 4-nél lehet)
- Mivel nincs alaplapi RTC csip, ezért a *bootboot.datetime* mindig 0000-00-00 00:00:00.
- SDHC Class 10-es kártyán kívül mással nem lett tesztelve
- Nem támogatja az AES-256-CBC enkriptált initrd-ket, csak az SHA-XOR-CBC-t.

Indítás

- **SD kártya**: másold le a *bootboot.img*-t ***FS0:\KERNEL8.IMG*** néven. Szükséged lesz egyéb firmver fájlokra (bootcode.bin, start.elf) is. A GPT-t nem ismeri, ezért az ESP-t le kell képezni az MBR-be hogy a Raspberry Pi firmver megtalálja a fájljait. A mellékelt *mkboot* program (https://gitlab.com/bztsrc/bootboot/blob/master/aarch64-rpi/mkboot.c) ezt megoldja neked.
- **Soros**: másold le a *bootboot.img*-t ***FS0:\KERNEL8.IMG*** néven, de ne csinálj BOOTBOOT könyvtárat. PC-n a *raspbootcom*-ra (https://gitlab.com/bztsrc/bootboot/blob/master/aarch64-rpi/raspbootcom.c), mrvn eredetijére, vagy az USBImager -S futtatására lesz szükség.

FÜGGELÉK

Egy GPT ESP partíció létrehozása

```
# fdisk /dev/sdc

Welcome to fdisk (util-linux 2.30.2).
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.

Device does not contain a recognized partition table.
Created a new DOS disklabel with disk identifier 0xfa00b86e.

Command (m for help): g
Created a new GPT disklabel (GUID: E6B4945A-8308-448B-9ACA-0E656854CF66).

Command (m for help): n p
Partition number (1-128, default 1): 1
First sector (2048-262110, default 2048):
Last sector, +sectors or +size{K,M,G,T,P} (2048-262110, default 262110): +8M

Created a new partition 1 of type 'Linux filesystem' and of size 8 MiB.

Command (m for help): t 1
Selected partition 1
Partition type (type L to list all types): 1
Changed type of partition 'Linux filesystem' to 'EFI System'.

Command (m for help): w
The partition table has been altered.
Syncing disks.
# mkfs.vfat -F 16 -n "EFI System" /dev/sdc1
mkfs.fat 4.1 (2017-01-24)
mkfs.fat: warning - lowercase labels might not work properly with DOS or Windows
# mkboot /dev/sdc
mkboot: GPT ESP mapped to MBR successfully
```

Egy minta BOOTBOOT kompatíbilis kernel

```
/*
 * mykernel/kernel.c
 *
 * Copyright (c) 2017 bzt (bztsrc@gitlab)
 *
 * Ez a fájl a BOOTBOOT Protokoll csomag része.
 * @brief Egy minta BOOTBOOT kompatíbilis kernel
 *
 */

/* sztring megjelenítése, lásd alább */
void puts(char *s);

/* nem feltételezzük, hogy van stdint.h */
typedef short int           int16_t;
typedef unsigned char       uint8_t;
typedef unsigned short int  uint16_t;
typedef unsigned int        uint32_t;
typedef unsigned long int   uint64_t;
```

```
#include <bootboot.h>

/* importált virtuális címek, lásd linker szkript */
extern BOOTBOOT bootboot;           // lásd bootboot.h
extern unsigned char *environment;  // környezet, UTF-8 szöveg kulcs=érték párok
extern uint8_t fb;                  // lineáris frémbuffer ide van leképezve

/*****************************************
 * Belépési pont, a BOOTBOOT betöltő hívja *
 *****************************************/
void _start()
{
    /*** FIGYELEM: ez a kód minden magon egyszerre, párhuzamosan fut ***/
    int x, y, s=bootboot.fb_scanline, w=bootboot.fb_width, h=bootboot.fb_height;

    // célkereszt, hogy lássuk jók-e a képernyőfelbontás dimenziói
    for(y=0;y<h;y++) { *((uint32_t*)(&fb + s*y + (w*2)))=0x00FFFFFF; }
    for(x=0;x<w;x++) { *((uint32_t*)(&fb + s*(h/2)+x*4))=0x00FFFFFF; }

    // rendre piros, zöld, kék négyzetek, hogy lássuk, jó-e a csatornasorrend
    for(y=0;y<20;y++) { for(x=0;x<20;x++) { *((uint32_t*)(&fb + s*(y+20) + (x+20)*4))=0x00FF0000; } }
    for(y=0;y<20;y++) { for(x=0;x<20;x++) { *((uint32_t*)(&fb + s*(y+20) + (x+50)*4))=0x0000FF00; } }
    for(y=0;y<20;y++) { for(x=0;x<20;x++) { *((uint32_t*)(&fb + s*(y+20) + (x+80)*4))=0x000000FF; } }

    // üdvözlés
    puts("Hello from a simple BOOTBOOT kernel");

    // kiakasztás, egyelőre nincs más
    while(1);
}

/******************************
 * Szöveg kiírása a képernyőre *
 ******************************/
typedef struct {
    uint32_t magic;
    uint32_t version;
    uint32_t headersize;
    uint32_t flags;
    uint32_t numglyph;
    uint32_t bytesperglyph;
    uint32_t height;
    uint32_t width;
    uint8_t glyphs;
} __attribute__((packed)) psf2_t;
extern volatile unsigned char _binary_font_psf_start;

void puts(char *s)
{
    psf2_t *font = (psf2_t*)&_binary_font_psf_start;
    int x,y,kx=0,line,mask,offs;
    int bpl=(font->width+7)/8;
    while(*s) {
        unsigned char *glyph = (unsigned char*)&_binary_font_psf_start + font->headersize +
            (*s>0&&*s<font->numglyph?*s:0)*font->bytesperglyph;
        offs = (kx * (font->width+1) * 4);
        for(y=0;y<font->height;y++) {
            line=offs; mask=1<<(font->width-1);
            for(x=0;x<font->width;x++) {
                *((uint32_t*)((uint64_t)&fb+line))=((int)*glyph) & (mask)?0xFFFFFF:0;
                mask>>=1; line+=4;
            }
            *((uint32_t*)((uint64_t)&fb+line))=0; glyph+=bpl; offs+=bootboot.fb_scanline;
        }
        s++; kx++;
    }
}
```

Egy minta Makefile

```
#
# mykernel/Makefile
#
# Copyright (c) 2017 bzt (bztsrc@gitlab)
#
# Ez a fájl a BOOTBOOT Protokoll csomag része.
# @brief Egy minta Makefile a példa kernelhez
#
#

CFLAGS = -Wall -fpic -ffreestanding -fno-stack-protector -nostdinc -nostdlib -I../

all: mykernel.x86_64.elf mykernel.aarch64.elf

mykernel.x86_64.elf: kernel.c
	x86_64-elf-gcc $(CFLAGS) -mno-red-zone -c kernel.c -o kernel.o
	x86_64-elf-ld -r -b binary -o font.o font.psf
	x86_64-elf-ld -nostdlib -nostartfiles -T link.ld kernel.o font.o -o mykernel.x86_64.elf
	x86_64-elf-strip -s -K mmio -K fb -K bootboot -K environment mykernel.x86_64.elf

mykernel.aarch64.elf: kernel.c
	aarch64-elf-gcc $(CFLAGS) -c kernel.c -o kernel.o
	aarch64-elf-ld -r -b binary -o font.o font.psf
	aarch64-elf-ld -nostdlib -nostartfiles -T link.ld kernel.o font.o -o mykernel.aarch64.elf
	aarch64-elf-strip -s -K mmio -K fb -K bootboot -K environment mykernel.aarch64.elf

clean:
	rm *.o *.elf *.txt
```

Egy minta linker szkript

```
/*
 * mykernel/link.ld
 *
 * Copyright (c) 2017 bzt (bztsrc@gitlab)
 *
 * Ez a fájl a BOOTBOOT Protokoll csomag része.
 * @brief Egy minta linker szkript a példa kernelhez
 *
 */

mmio  = 0xfffffffff8000000;
fb    = 0xfffffffffc000000;
PHDRS {
    boot PT_LOAD;                               /* egy betölthető szegmens */
}
SECTIONS
{
    . = 0xffffffffffe00000;
    bootboot    = .; . += 4096;
    environment = .; . += 4096;
    .text : {
        KEEP(*(.text.boot)) *(.text .text.*)    /* kód */
        *(.rodata .rodata.*)                    /* adat */
        *(.data .data.*)
    } :boot
    .bss (NOLOAD) : {                           /* bss */
        . = ALIGN(16);
        *(.bss .bss.*)
        *(COMMON)
    } :boot
}
```

Egy minta többprocesszoros (SMP) inicializáló kód

x86_64*

```
_start:
  mov  $1, %eax
  cpuid
  shr  $24, %ebx
  cmpw %bx, bootboot + 0xC   // bootboot.bspid
  jne  .ap
  /* alapprocesszoron futtatandó dolgok */
.ap:
  /* kiszolgálóprocesszorokon futtatandó dolgok */

(* - APIC csak 256 magot tud kezelni. Használj x2APIC-ot a teljes kompatibilitáshoz, egészen 65536
magig)
```

AArch64

```
_start:
  mrs  x0, mpidr_el1
  and  x0, x0, #3
  cbnz x0, .ap     // BSP mindig a 0-ás
  /* alapprocesszoron futtatandó dolgok */
.ap:
  /* kiszolgálóprocesszorokon futtatandó dolgok */
```

Egy minta grub.cfg bejegyzés

```
menuentry "MyKernel" {
    multiboot /bootboot/loader      # bootboot.bin
    module /bootboot/initrd         # az első modul az initrd (memórialemezkép, opcionális)
    module /bootboot/config         # a második modul az environment fájl (környezet. opcionális)
    boot
}
```

SZÓJEGYZÉK