

# Proces uruchamiania FreeBSD przez bootloader U-Boot

Rafał Czubak  
rcz@semihalf.com

AGH, Kraków 21.05.2009

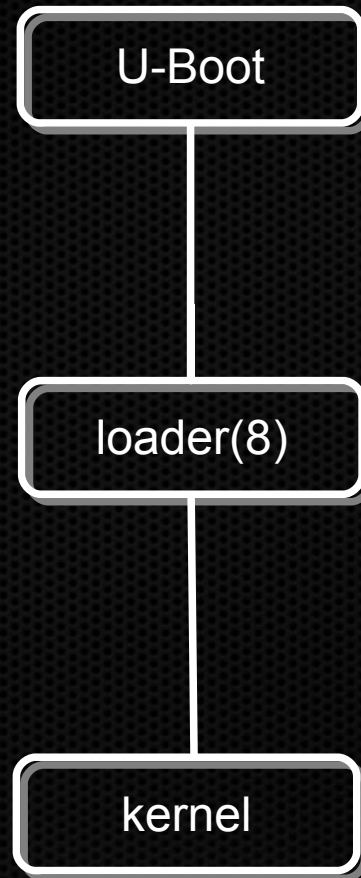
# Szkic prezentacji

- Wstęp
- U-Boot
  - Szczegóły działania i funkcjonalność
  - API dla samodzielnych aplikacji
- loader(8)
  - Budowa i działanie
  - Interfejs z firmware'm
  - Uruchamianie kernela i przekazywanie metadanych
- Kernel FreeBSD w momencie startu
- Podsumowanie, pytania, komentarze

# Uruchamianie FreeBSD

- Trzy etapy bootowania
  - Bootloader pierwszego poziomu – firmware
  - Bootloader ostatniego poziomu – loader(8)
  - Kernel
- Dalsze uruchamianie systemu
  - Zamontowanie root filesystemu
  - Uruchomienie /sbin/init
  - Tryb pojedynczego użytkownika
  - Tryb wielu użytkowników
- Na przykładzie produktów firmy Marvell

# Uruchamianie FreeBSD



# U-Boot

- Firmware
  - Przechowywany w pamięciach nieulotnych
  - Kod uruchamiany przy starcie maszyny
  - Inicjalizuje sprzęt
  - Uruchamia samodzielną aplikację (loader)
- Dodatkowa funkcjonalność

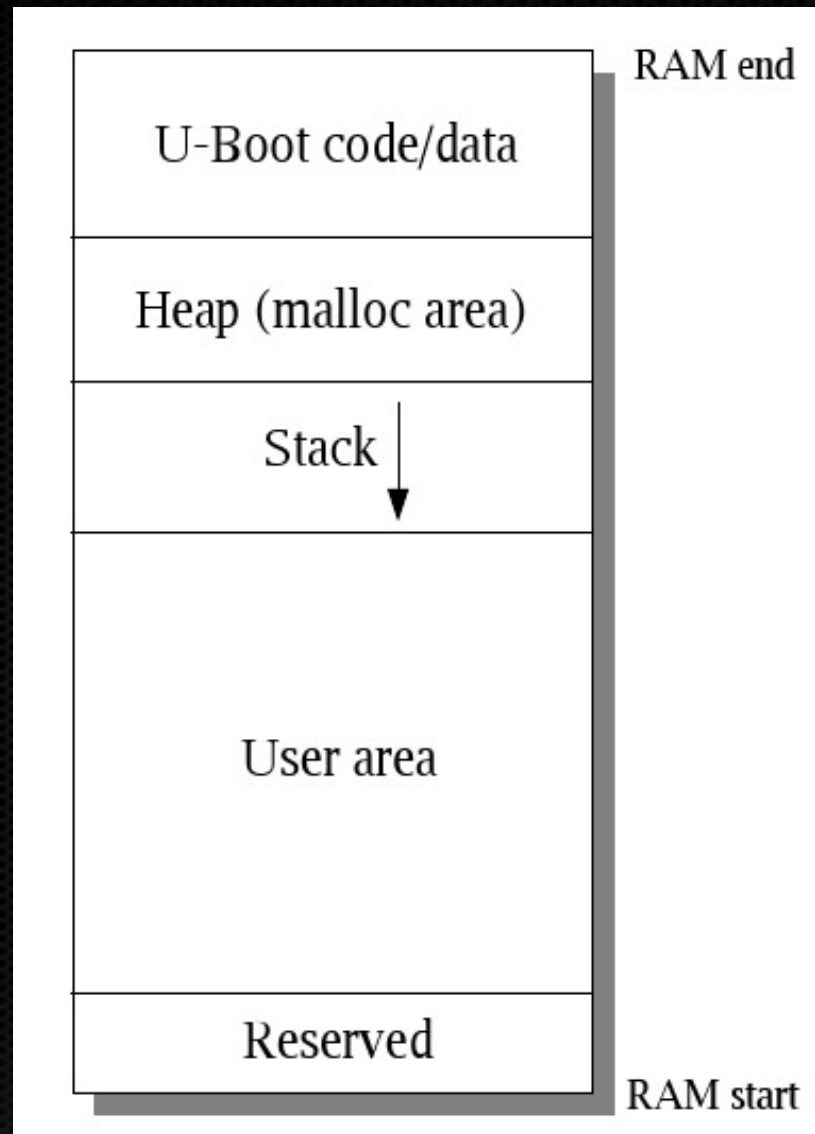
# U-Boot – podsumowanie

- Kod źródłowy napisany w C i assemblerze
- Własne drzewo z systemem buildów
- Ściśle związany z Linuxem
  - Interfejsy (np. FDT)
  - Styl kodowania
- Wsparcie dla wielu architektur i platform
- Wsparcie dla wielu systemów operacyjnych
- Licencja GPLv2
- Dawniej: PPCBoot

# U-Boot – podsumowanie c.d.

- Szczegóły działania
  - Uruchomienie z pamięci nieulotnej
  - Inicjalizacja CPU, pamięci cache, założenie stosu
  - Uaktywnienie konsoli
  - Inicjalizacja kontrolera pamięci RAM
  - Relokacja do pamięci RAM
  - Dalsze wykonywanie kodu z pamięci RAM
    - Inicjalizacja peryferiów
    - Automatyczne bootowanie
    - Tryb interaktywny (wiersz poleceń)

# U-Boot w pamięci RAM



# U-Boot – warianty uruchamiania

- Bootowanie kernela/aplikacji
  - ELF – bootelf
  - Proste binarium – go
  - Entry point – różnice między prostym binarium a ELF
- Bootowanie przez bootm
  - mkimage
  - Różne rodzaje bootowanych obrazów
  - Obrazy mogą być skompresowane

# U-Boot – funkcjonalność

- Bootowanie systemu operacyjnego z różnych nośników
  - Pamięci Flash (NAND, NOR)
  - Sieć (netbooting)
  - Urządzenia pamięci masowej (S/ATA, SCSI, USB)
  - Obsługa różnych systemów plików (EXT2, FAT, JFFS2)
- Operacje w pamięci RAM
  - Podgląd (md)
  - Zmiana zawartości (mw, mm, nm, cp)
  - Diagnostyka (cmp, crc32, mtest)
- Enumeracja urządzeń

# U-Boot – funkcjonalność c.d.

- Zmienne środowiskowe
  - Przechowywanie ustawień
  - Możliwość definiowania własnych „skryptów”
- Podstawowe wsparcie dla sieci
  - ping, CDP
  - BOOTP, DHCP, RARP
  - TFTP
  - NFS
  - SNTP
- Automatyczna aktualizacja
  - Własnego obrazu
  - Kernela lub samodzielnych aplikacji

# U-Boot – przykład uruchomienia

U-Boot 1.1.4 (Dec 18 2008 - 11:45:09) Semihalf (Marvell version: 3.4.8)

U-Boot code: 00600000 -> 0067FFF0 BSS: -> 00693548

Soc: 88F6281 A0 (DDR2)

CPU running @ 1200Mhz L2 running @ 400Mhz

SysClock = 400Mhz , TClock = 200Mhz

DRAM CAS Latency = 5 tRP = 5 tRAS = 18 tRCD=6

DRAM CS[0] base 0x00000000 size 256MB

DRAM CS[1] base 0x10000000 size 256MB

DRAM Total size 512MB 16bit width

Flash: 0 kB

Addresses 8M - 0M are saved for the U-Boot usage.

Mem malloc Initialization (8M - 7M): Done

NAND:128 MB

CPU : Marvell Feroceon (Rev 1)

Streaming enabled

Write allocate enabled

USB 0: host mode

PEX 0: interface detected no Link.

Net: egiga0 [PRIME]

Hit any key to stop autoboot: 0

Marvell>>

# U-Boot – API

- Udostępnia funkcjonalność U-Boot
- Wywołanie funkcji – syscall
- Aplikacja dostarcza kontekstu i argumentów
- Stos U-Bootowy
- Sterta inicjalizowana w aplikacji
- Niezależne od architektury

# U-Boot – omówienie API

- Pozwala na:
  - Pobranie/wysłanie znaku z konsoli/ na konsolę
  - Enumerację urządzeń
  - Otwieranie, zamykanie urządzeń
  - Odczyt/zapis z/do urządzenia
  - Wysłanie/odebranie pakietu
  - Enumerację, pobranie i ustawienie zmiennych środowiskowych
  - Reset systemu, wyświetlenie informacji o systemie
  - Wygenerowanie opóźnienia, pobranie czasu

# U-Boot – inicjalizacja API

```
void api_init(void)
{
    struct api_signature *sig = NULL;

    /* TODO put this into linker set one day... */
    calls_table[API_RSVD] = NULL;
    calls_table[API_GETC] = &API_getc;
    calls_table[API_PUTC] = &API_putc;
    calls_table[API_TSTC] = &API_tstc;
    calls_table[API_PUTS] = &API_puts;
    calls_table[API_RESET] = &API_reset;
    calls_table[API_GET_SYS_INFO] = &API_get_sys_info;
    calls_table[API_UDELAY] = &API_udelay;
    calls_table[API_GET_TIMER] = &API_get_timer;
    calls_table[API_DEV_ENUM] = &API_dev_enum;
    calls_table[API_DEV_OPEN] = &API_dev_open;
    calls_table[API_DEV_CLOSE] = &API_dev_close;
    calls_table[API_DEV_READ] = &API_dev_read;
    calls_table[API_DEV_WRITE] = &API_dev_write;
    calls_table[API_ENV_GET] = &API_env_get;
    calls_table[API_ENV_SET] = &API_env_set;
    calls_table[API_ENV_ENUM] = &API_env_enum;
    calls_no = API_MAXCALL;

    debugf("API initialized with %d calls\n", calls_no);

    dev_stor_init();
}
```

# U-Boot – sygnatura API

```
struct api_signature {  
    char        magic[API_SIG_MAGLEN];    /* magic string */  
    uint16_t    version;                  /* API version */  
    uint32_t    checksum;                  /* checksum of this sig struct */  
    scp_t       syscall;                   /* entry point to the API */  
};
```

# U-Boot – inicjalizacja sygnatury API

```
/*
 * Produce the signature so the API consumers can find it
 */
sig = malloc(sizeof(struct api_signature));
if (sig == NULL) {
    printf("API: could not allocate memory for the signature!\n");
    return;
}

debugf("API sig @ 0x%08x\n", sig);
memcpy(sig->magic, API_SIG_MAGIC, 8);
sig->version = API_SIG_VERSION;
sig->syscall = &syscall;
sig->checksum = 0;
sig->checksum = crc32(0, (unsigned char *)sig,
                    sizeof(struct api_signature));
debugf("syscall entry: 0x%08x\n", sig->syscall);
}
```

# U-Boot – syscall API

```
int syscall(int call, int *retval, ...)
{
    va_list ap;
    int rv;

    if (call < 0 || call >= calls_no) {
        debugf("invalid call #%d\n", call);
        return 0;
    }

    if (calls_table[call] == NULL) {
        debugf("syscall #%d does not have a handler\n", call);
        return 0;
    }

    va_start(ap, retval);
    rv = calls_table[call](ap);
    if (retval != NULL)
        *retval = rv;

    return 1;
}
```

# U-Boot – przykładowa funkcja API

```
static int API_putc(va_list ap)
{
    char *c;

    if ((c = (char *)va_arg(ap, u_int32_t)) == NULL)
        return API_EINVAL;

    putc(*c);
    return 0;
}
```

# U-Boot, API demo

## Pokaz praktyczny

# loader(8)

- Natywny bootloader FreeBSD
- Kod zależny od platformy
- Kod niezależny od platformy
- Oparty na bibliotece libstand(3)
  - Uproszczone libc
    - printf(), malloc(), free()
    - Wsparcie dla protokołów sieciowych
    - Wsparcie dla systemów plików
- Podstawowe operacje zależne od firmware'u
- Interfejs użytkownika niezależny od sprzętu

# loader(8)

- Interfejs z firmwarem
  - Korzysta z U-Bootowego API
    - Wrapper na syscall
  - Szuka sygnatury API w pamięci
    - Rejestr SP – hint
  - Otrzymuje listę urządzeń od U-Boota

# loader(8) – start dla architektury ARM

```
/*
 * Entry point to the loader that U-Boot passes control to.
 */
    .text
    .globl _start
_start:
    /* Hint where to look for the API signature */
    ldr    ip, =uboot_address
    str    sp, [ip]

    /* Save U-Boot's r8 */
    ldr    ip, =saved_regs
    str    r8, [ip, #0]

    /* Start loader */
    b     main
```

# loader(8) – start dla architektury PowerPC

```
/*
 * Entry point to the loader that U-Boot passes control to.
 */
        .text
        .globl _start
_start:
        /* Hint where to look for the API signature */
        lis     %r11, uboot_address@ha
        addi   %r11, %r11, uboot_address@l
        stw   %r1, 0(%r11)
        /* Save U-Boot's r14 */
        lis     %r11, saved_regs@ha
        addi   %r11, %r11, saved_regs@l
        stw   %r14, 0(%r11)
        /* Disable interrupts */
        mfmsr  %r11
        andi.  %r11, %r11, ~0x8000@l
        mtmsr  %r11
        b     main
```

# loader(8) – poszukiwanie sygnatury API

```
int
api_search_sig(struct api_signature **sig)
{
    unsigned char *sp, *spend;

    if (sig == NULL)
        return (0);

    if (uboot_address == 0)
        uboot_address = 255 * 1024 * 1024;

    sp = (void *)(uboot_address & ~0x000fffff);
    spend = sp + 0x00300000 - API_SIG_MAGLEN;
    while (sp < spend) {
        if (!bcmp(sp, API_SIG_MAGIC, API_SIG_MAGLEN)) {
            *sig = (struct api_signature *)sp;
            if (valid_sig(*sig))
                return (1);
        }
        sp += API_SIG_MAGLEN;
    }

    *sig = NULL;
    return (0);
}
```

# loader(8) – syscall, wykonanie wołania

```
/*
 * syscall()
 */
ENTRY(syscall)
    /* Save caller's lr */
    ldr    ip, =saved_regs
    str    lr, [ip, #4]
    /* Save loader's r8 */
    ldr    ip, =saved_regs
    str    r8, [ip, #8]

    /* Restore U-Boot's r8 */
    ldr    ip, =saved_regs
    ldr    r8, [ip, #0]
    /* Call into U-Boot */
    ldr    lr, =return_from_syscall
    ldr    ip, =syscall_ptr
    ldr    pc, [ip]
```

# loader(8) – syscall, powrót z wołania

```
return_from_syscall:
    /* Restore loader's r8 */
    ldr    ip, =saved_regs
    ldr    r8, [ip, #8]
    /* Restore caller's lr */
    ldr    ip, =saved_regs
    ldr    lr, [ip, #4]
    /* Return to caller */
    mov   pc, lr
```

# loader(8) – wywołanie funkcji API

```
void
ub_putc(char c)
{
    syscall(API_PUTC, NULL, (uint32_t)&c);
}
```

# loader(8) – lista wołań

```
enum {  
    API_RSVD = 0,  
    API_GETC,  
    API_PUTC,  
    API_TSTC,  
    API_PUTS,  
    API_RESET,  
    API_GET_SYS_INFO,  
    API_UDELAY,  
    API_GET_TIMER,  
    API_DEV_ENUM,  
    API_DEV_OPEN,  
    API_DEV_CLOSE,  
    API_DEV_READ,  
    API_DEV_WRITE,  
    API_ENV_ENUM,  
    API_ENV_GET,  
    API_ENV_SET,  
    API_MAXCALL  
};
```

# loader(8) – przykład uruchomienia

```
Marvell>> bootelf
Loading .text @ 0x01000074 (123568 bytes)
Loading .rodata @ 0x0101e324 (2912 bytes)
Loading .rodata.str1.4 @ 0x0101ee84 (16387 bytes)
Loading set_Xcommand_set @ 0x01022e88 (84 bytes)
Loading .data @ 0x01023000 (6300 bytes)
Clearing .bss @ 0x0102489c (7364 bytes)
## Starting application at 0x0100009c ...
Consoles: U-Boot console
Compatible API signature found @721968
Number of U-Boot devices: 2

FreeBSD/arm U-Boot loader, Revision 1.0
(rcz@semihalf.com, Wed May 20 16:05:52 CEST 2009)
DRAM:      512MB

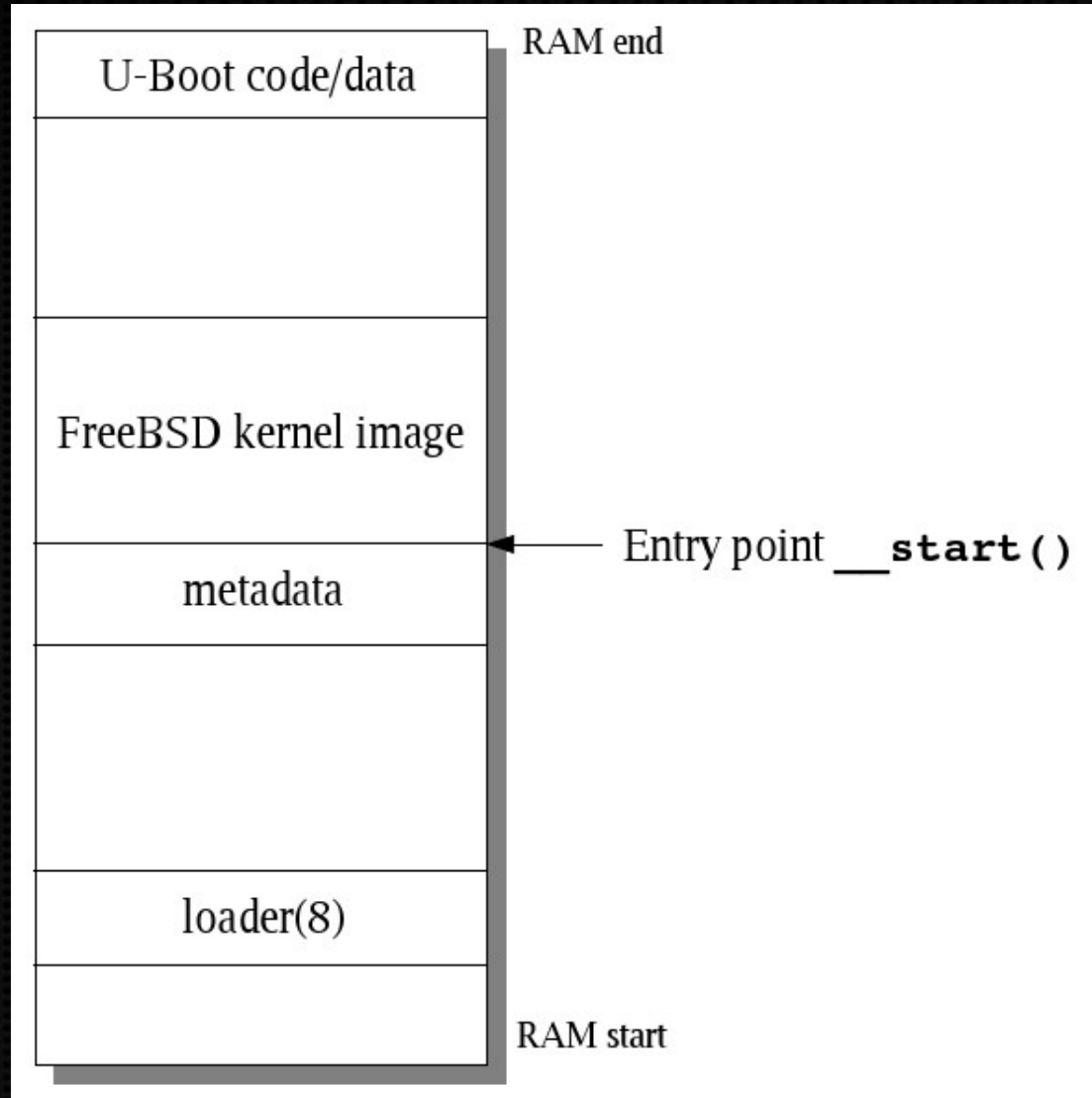
can't load 'kernel'

Type '?' for a list of commands, 'help' for more detailed help.
loader>
```

# loader(8) – podsumowanie

- Pozwala na ładowanie modułów
- Przekazuje kernelowi metadane
- Uruchamia kernel

# loader(8), kernel – zależności



# Kernel podczas startu

- Przechwycenie wskaźnika do metadanych
- Odczytanie metadanych
  - bootinfo
  - boothowto
  - Regiony pamięci
- Brak metadanych – wartości domyślne w ramach platformy

# Kernel: uruchomienie

```
loader> boot /boot/kernel/kernel-6x -a
/boot/kernel/kernel-6x-hal text=0x327e16 data=0x24abc+0x18d840
syms=[0x4+0x761d0+0x4+0x578d8]
Kernel entry at 0x900100 ...
KDB: debugger backends: ddb
KDB: current backend: ddb
Copyright (c) 1992-2009 The FreeBSD Project.
Copyright (c) 1979, 1980, 1983, 1986, 1988, 1989, 1991, 1992, 1993, 1994
    The Regents of the University of California. All rights reserved.
FreeBSD is a registered trademark of The FreeBSD Foundation.
FreeBSD 8.0-CURRENT #0: Wed Mar 25 23:57:32 CET 2009
WARNING: WITNESS option enabled, expect reduced performance.
WARNING: DIAGNOSTIC option enabled, expect reduced performance.
CPU: Feroceon 88FR131 rev 1 (write-through core)
    WB enabled EABT branch prediction enabled
    16KB/32B 4-way Instruction cache
    16KB/32B 4-way write-back-locking-C Data cache
real memory = 536870912 (512 MB)
avail memory = 517955584 (493 MB)
SOC: Marvell 88F6281 rev Z0, TClock 166MHz
mbus0: <Marvell Internal Bus (Mbus)>
ic0: <Marvell Integrated Interrupt Controller> at mem 0xf1020200-0xf102023b on mbus0
(...)
```

# Uruchamianie kernela

## Pokaz praktyczny

# Podsumowanie

- U-Boot
  - API
- loader(8)
- Kernel FreeBSD w momencie startu

# Bibliografia, podziękowania

- The FreeBSD Project

- FreeBSD resources

<http://www.freebsd.org/doc/en/books/handbook/>

<http://www.freebsd.org/doc/en/books/arch-handbook/>

<http://www.freebsd.org/doc/en/books/developers-handbook/>

- Zespół programistów Semihalf

Dziękuję za uwagę!

Pytania, komentarze?

# Proces uruchamiania FreeBSD przez bootloader U-Boot

Rafał Czubak  
rcz@semihalf.com

AGH, Kraków 21.05.2009