
Univerzitetni študijski program, 3. letnik

# Sistemska programska oprema

predavatelj: doc. Tomaž Dobravec

## Nalagalnik

# Nalagalnik

**Osnovna naloga nalagalnika**

- Nalagalnik naloži program v pomnilnik in poskrbi, da se program začne izvajati.

**Vrste nalagalnikov**

- Absolutni nalagalnik
  Nalaganje na vnaprej določen naslov
- Nalagalnik s prenaslavljanjem
  Nalaganje na naslov, ki ga pridobi od OS + popravljanje neposrednih naslovov
- Dinamični nalagalnik
  Nalaganje v času izvajanja

# Absolutni nalagalnik

- Absolutni nalagalnik naloži absolutne programe
- Nalagalni naslov absolutnih programov je za vse kontrolne sekcije določen v času pisanja programa.
- Do konca so razrešeni vsi naslovi.

**Naloga absolutnega nalagalnika je, da**

- prebere objektno datoteko,
- pretvori morebitne tekstovne zapise v binarne,
- objektno kodo naloži na predvidene lokacije in
- začne izvajati program.

# Absolutni nalagalnik - primer

```
HCOPY  001000001007A
T001000 1E 14103348 20390010 36281030 30101548 20613C10 0300102A 0C103900 102D0C10
T00101E 15 0C103648 20610810 334C0000 454F4600 00030000 00
T002039 1E 04103000 1030E020 5D30203F D8205D28 10303020 57549039 2C205E38 203F
T002057 1C 1010364C 0000F100 10000041 030E0207 93020645 09039DC2 0792C1036
T002073 07 3820644C 000005
E001000
```

(a) Object program

| Memory address | Contents | | | | |
|---|---|---|---|---|---|
| 0000 | xxxxxxxx | xxxxxxxx | xxxxxxxx | xxxxxxxx | |
| 0010 | xxxxxxxx | xxxxxxxx | xxxxxxxx | xxxxxxxx | |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | |
| 0FF0 | xxxxxxxx | xxxxxxxx | xxxxxxxx | xxxxxxxx | |
| 1000 | 14103348 | 20390010 | 36281030 | 30101548 | |
| 1010 | 20613C10 | 0300102A | 0C103900 | 102D0C10 | |
| 1020 | 36482061 | 0810334C | 0000454F | 46000003 | |
| 1030 | 000000xx | xxxxxxxx | xxxxxxxx | xxxxxxxx | ←COPY |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | |
| 2030 | xxxxxxxx | xxxxxxxx | xx041030 | 001030E0 | |
| 2040 | 205D3020 | 3FD8205D | 28103030 | 20575490 | |
| 2050 | 392C205E | 38203F10 | 10364C00 | 00F10010 | |
| 2060 | 00041030 | E0207930 | 20645090 | 39DC2079 | |
| 2070 | 2C103638 | 20644C00 | 0005xxxx | xxxxxxxx | |
| 2080 | xxxxxxxx | xxxxxxxx | xxxxxxxx | xxxxxxxx | |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | |

(b) Program loaded in memory

# Nalaganje s prenaslavljanjem

Nalaganje s prenaslavljanjem uporabimo takrat, ko nalagamo prenaslovljiv modul.

Naloga nalagalnika je, da

1. od operacijskega sistema pridobi nalagalni naslov

2. prebere objektni modul in ga naloži na pravilni naslov; naslov objektne kode izračuna po formuli

naslov kode = originalni naslov kode + nalagalni nalsov - naslov

T 00001D 0D 000003….
- nalgalni naslov = 0x0A100
T 00A11D
1D + A100 = A11D

# Nalaganje s prenaslavljanjem

3. popravi dele programske kode v skladu z navodili, ki so zapisana v prilagoditvenih zapisih; pri tem primerno prilagaja naslove podane v modifikacijskih zapisih.

M 001061 05 + COPY(A100) - kam se je program nalozu
- nalagalni nalov = 0xA100
M 00B161 05 + A100
1061 + A100 = B161

# Nalaganje s prenaslavljanjem

4. izvajanje programa preusmeri na naslov, podan v E zapisu; pri tem podanemu naslovu prišteje nalagalni naslov

E 000000
E 00A100 - nalagalni naslov

# Nalaganje s prenaslavljanjem - primer

- nalagalni naslov 0xA100

Prenaslovljiv program `COPY` "naloži" v pomnilnik na lokacijo `0x0A100`.

```
HCOPY 000000 00107A
T 000000 1D 172027 4B101033 032023 …
T 00001D 0D 000003 0f200A 4B10105E …
   …
T 001033 1D B410 B400 …
T 001050 0E 3B2FE9 13100000 4F0000 F1 001000
T 00105E 1C B410 7710002D E32012 …
E 000000

M 000004 05 +COPY
M 000024 05 +COPY
M 001061 05 +COPY
```

M 000A104 + A100
M 000A124 + A100
M 000B161 + A100

| Naslov | Vsebina | | | |
|---|---|---|---|---|
| 0000 | xxxxxxxx | xxxxxxxx | xxxxxxxx | xxxxxxxx |
| … | ……. | ……. | ……. | ……. |
| A0F0 | xxxxxxxx | xxxxxxxx | xxxxxxxx | xxxxxxxx |
| A100 | ……. | ……. | ……. | ……. |
| | ……. | ……. | ……. | ……. |
| | ……. | ……. | ……. | ……. |
| | ……. | ……. | ……. | ……. |
| | ……. | ……. | ……. | ……. |
| | ……. | ……. | ……. | ……. |
| | ……. | ……. | ……. | ……. |
| | ……. | ……. | ……. | ……. |
| | ……. | ……. | ……. | ……. |
| … | ……. | ……. | ……. | ……. |

# Dinamični nalagalnik

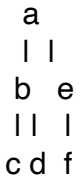
Kako ravnati v primeru, ko je velikost programa večja od razpoložljivega pomnilnika?

```
A() {
  B();
  E();
}
```

```
B() {
  C();
  D();
}
```

```
E() {
  F();
}
```

a) Uporaba navideznega pomnilnika
   - potrebujemo podporo OS
   - izvede se ostranjevanje ali segmentacija




b) Dinamično nalaganje
   - hkrati so v pomnilniku le medseboj odvisni deli programa
   - neodvisni deli programa lahko zasedejo ISTO pomnilniško lokacijo

# Dinamični nalagalnik

## Naloge prevajalnika

- detektira neodvisne dele kode,
- namesto klicev prekrivajočih delov programa vključi klic dinamičnega nalagalnika

## Naloga povezovalnika

- če dinamičen nalagalnik ni del OSa, mora povezovalnik kodo dinamičnega nalaganja vključiti v kodo prevedenega programa.

## Naloge dinamičnega nalagalnika:

- naloži del kode na predvideno mesto,
- popravi morebitne neposredne naslove,
- kliče dinamično naloženo kodo.

# Začetni nalagalnik

Kaj se zgodi, ko se računalnik "zbudi"?

- Načeloma: ob zagonu je računalnik prost (idle) brez naloženih programov v pomnilniku.
- Kdo potemtakem sproži začetek vsega dogajanja (prvo nalaganje)?

Kje je začetek?

- Kdo naloži nalagalnik?
- Kdo pa naloži OS?
- Kdo pa naloži nalagalnik za OS?
- Kdo pa naloži nalagalnik za nalagalnik za OS?

# Začetni nalagalnik

- Da celoten postopek nalaganja sploh začne teči, potrebujemo preprost nalagalnik, ki naloži prvi program.
- Temu nalagalniku rečemo začetni nalagalnik (angl. *bootstrap loader*).
- Začetni nalagalnik poskrbi, da se začne večfazni postopek začetnega nalaganja. Pri tem se v pomnilnik drug za drugim vnašajo pomožni (vedno bolj kompleksni) nalagalniki, ki prevzamejo nadaljne naloge.

# Začetni nalagalnik

Osnovno vprašanje ostaja: Kdo naloži **začetni** nalagalnik?

V praksi se pojavljajo tri rešitve:

a) Ob zagonu operater z uporabo prilagojene strojne opreme vnese prve ukaze
   - zgodovina
   - lahko pride do napak

b) Celoten začetni nalagalnik je shranjen v ROMu; ob zagonu računalnika se koda prepiše v RAM in izvajanje se prenese tja
   - slabost: začetnega nalagalnika ne moremo spreminjati

# Začetni nalagalnik

c) Začetni nalagalnik je sestavni del interne procesorjeve logike izveden kot poseben ukaz ali zelo kratek program v ROMu

- Izvajanje tega ukaza oziroma tega programa iz ROMa povzroči branje programske kode **vnaprej določene dolžine iz vnaprej določene naprave na vnaprej določen naslov v pomnilniku**,
- po branju se izvajanje prenese na prebrano kodo.

- Običajno se nalagalniki nalagajo postopoma: zelo preprost nalagalnik naloži malo bolj kompleksen nalagalni, ta naloži še kompleksnejšega, …

# Začetno nalaganje pri Intelovih procesorjih

- Pri Intelovih procesorjih (od i80286 naprej) se zgornjih 64K naslovnega prostora (pod 1MB) imenuje **sistemsko inicializacijsko področje** (SIP)

- v SIP se v postavi stalni pomnilnik z
  - zagonskimi in diagnostičnimi funkcijami (POST) ter
  - osnovni vhodno-izhodni sistem (BIOS).

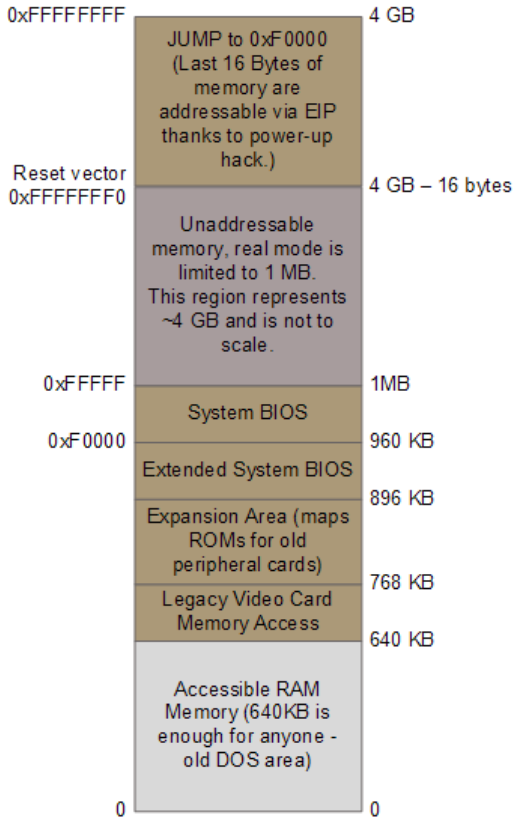
- ob zagonu računalnika:

```
PC := NAJVIŠJI_NASLOV - 16
```

  - 0xFFFFFFFF – 0x00000010 = 0xFFFFFFF0

na tem naslovu je zapisan ukaz za skok SIP - najprej se izvede POST nato pa se s pomočjo prekinitve 0x19 sproži osnovno začetno nalaganje.

Podrobneje: http://duartes.org/gustavo/blog/post/how-computers-boot-up



# Glavni nalagalni sektor

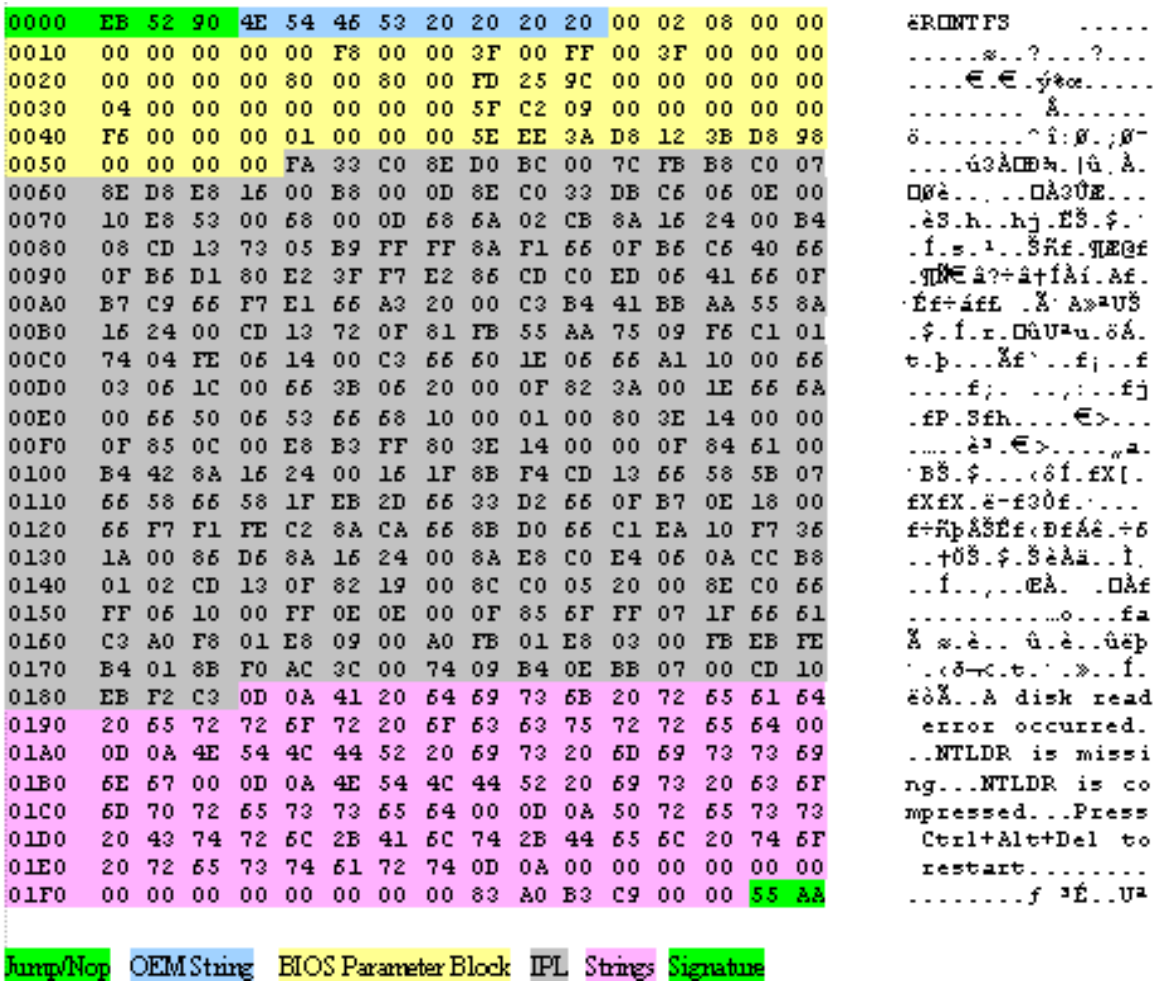
- BIOS omogoča le fizičen (ne pa tudi logičnega) dostop do zunanjih enot;
- začetnega nalagalnika zato ne moremo prebrati iz datoteke;
- začetni nalagalnik se nahaja v vnaprej določenem sektorju posamezne enote;
- pri osebnih računalnikih se kot stalni sektor uporablja prvi sektor na enoti. Imenuje se glavni nalagalni sektor ali MBR (master boot record).

# Glavni nalagalni sektor

V MBR so poleg programske kode, ki sproži postopek nalaganja, shranjene tudi naslednje informacije:

- dopolnilni program za izbiro operacijskega sistema (opcijsko),
- podatki o lastnostih nalagalne naprave oziroma BIOSov blok parametrov (BIOS parameter block, BPB),
- informacije o particijah naprave.

# Glavni nalagalni sektor



# Glavni nalagalni sektor

BPB vsebuje naslednje podatke (primer za FAT32):

| Pomen | Odmik | Dolžina |
|---|---|---|
| Oznaka OS | 3 | 8 |
| Število zlogov na sektor | 11 | 2 |
| Število sektorjev v gruči | 13 | 1 |
| Rezervirani sektorji | 14 | 2 |
| Število FAT | 16 | 1 |
| Število korenskih vstopov v kazalu | 17 | 2 |
| Število majhnih sektorje | 19 | 2 |
| Opis medija | 21 | 1 |
| Število sektorjev FAT | 22 | 2 |
| Število sektorjev na stezi | 24 | 2 |
| Število glav | 26 | 2 |
| Število skritih sektorjev | 28 | 4 |
| Število velikih sektorjev | 32 | 4 |

več: http://en.wikipedia.org/wiki/BIOS_parameter_block

# Glavni nalagalni sektor

**Kako poteka nalaganje s pomočjo MBR?**

- uporabnik v BIOS nastavi vrstni red nalagalnih enot
- po izvedbi POST se v BIOSu kliče funkcija (interrupt) 0x19, ki povzroči branje prvega sektorja s prve prisotne nalagalne enote;
- vsebina sektorja se prenese na naslov 0x0000:7C00, izvajanje se preusmeri na 0000:7C00
- če na izbrani nalagalni enoti ni operacijskega sistema,
  - koda v MBR izpiše sporočilo "Non-system disk or disk eror",
- sicer
  - aktivira se rutina za izbor operacijskega sistema (npr. LILO ali GRUB), katere osnovni del je v MBR.
- z izbiro OSa uporabnik določi particijo, iz katere se v nadaljevanju prebere BR (boot record); MBR prebere ta BR na naslov 0x0000:7C00 in izvajanje preseli nanj; koda tega BRja poskrbi za nalaganja OSa.

# Glavni nalagalni sektor

Branje in prikaz MBR v Linux sistemih

- preberi MBR na svojem računalniku

```
dd if=/dev/ime_diska of=backup.mbr bs=512 count=1
```

- prikaži MBR v hex obliki

```
hexdump -C backup.mbr
```

- na podlagi podatkov iz BPB izračunaj število sektorjev na disku

- poglej, kaj izpiše program `file`

```
file backup.mbr
```

# Glavni nalagalni sektor

## Programska koda prebranega MBRja (Linux, FAT32)

```
00007C5A  FA              cli
00007C5B  31C0            xor ax,ax
00007C5D  8ED0            mov ss,ax
00007C5F  BC007C          mov sp,0x7c00
00007C62  FB              sti
00007C63  8ED8            mov ds,ax
00007C65  E80000          call 0x7c68         //push PC
00007C68  5E              pop si              //si=PC
00007C69  83C619          add si,byte +0x19   //si=si+0x19
00007C6C  BB0700          mov bx,0x7
00007C6F  FC              cld
00007C70  AC              lodsb               // v al da en bajt iz sp:si
00007C71  84C0            test al,al          // konec niza?
00007C73  7406            jz 0x7c7b
00007C75  B40E            mov ah,0xe
00007C77  CD10            int 0x10            // izpiše znak
00007C79  EBF5            jmp short 0x7c70    // ponovi
00007C7B  30E4            xor ah,ah
00007C7D  CD16            int 0x16            // read keyboard
00007C7F  CD19            int 0x19            // boot strap
```