# Uvod u kernel

*Matko Kosmat | 30 ožujka, 2026*

Kao uvod u seriju članaka o Linux jezgri, u nastavku ćemo dati namjerno površan pregled što je uopće jezgra (engl. kernel), čemu služi te koji su osnovni tipovi jezgri operacijskog sustava. Zbog jednostavnosti i kratkoće pregleda, tek su letimično spomenuti različiti složeni koncepti dizajna operacijskog sustava. Zainteresiranom čitatelju preporučamo čitanje odgovarajuće literature, počevši sa knjigom Andrew Tanenbaum: "Operating Systems - Design and Implementation".

Operacijski sustav je specijalan program ili skupina programa koji upravlja računalom u smislu sklopovlja (engl. hardware) i programske podrške (engl. software). Operacijski sustav tvori osnovnu podršku za različite dodatne programe odnosno softver, pružajući različite usluge. Takve usluge su na primjer upravljanje procesima, upravljanje memorijom, upravljanje datotečnim sustavima, mrežna podrška, upravljanje sigurnošću, grafičko ili tekstualno sučelje, upravljački programi za specifično sklopovlje i sl. Operacijski sustavi se razlikuju po namjeni (osobna računala, ugrađeni sustavi, mainframe računala), filozofiji rada i izvedbe (DOS, Unix, Windows, OS X), programskom jeziku u kojem je implementiran, itd. Kao rezultat takvih različitih kriterija namjene i izvedbe danas postoje stotine različitih operacijskih sustava.

Jezgra operacijskog sustava je centralni dio modernog operacijskog sustava. Njena zadaća je upravljanje sklopovljem na najnižem mogućem nivou kao i pružanje različitih već navedenih usluga aplikacijama. Jezgra upravlja računalnim resursima i predstavlja sloj između samih korisničkih programa odnosno korisničke okoline (engl. user land) i fizičkog računalnog sklopovlja. Takav sloj nazivamo i jezgrinom okolinom (engl. kernel land) i on može biti "deblji" ili "tanji" u ovisnosti o količini usluga koje jezgra pruža korisničkoj okolini. Razlog postojanja takvog sloja je u dizajnu većine modernih operacijskih sustava koji podrazumijeva niz apstrakcija koje se oslanjanju jedna na drugu. Takav dizajn se najčešće promatra kao piramida apstrakcija ili kao niz kružnih slojeva pri čemu je jezgra najdublje i predstavlja sloj najbliži samom sklopovlju računala. Kao posljedica ovakvog načina izvedbe operacijskog sustava dobivamo pojednostavljenje korisničkih aplikacija budući da one ne moraju znati specifičnosti za pojedino računalo odnosno sklopovlje čije usluge koriste. Aplikacije se mogu izvršavati na različitom sklopovlju i dobivaju uniformnu apstrakciju, a jezgra se brine o različitim internim aspektima.

Tipična jezgra modernog operacijskog sustava se tipično izvršava u tzv. nadglednom načinu rada (engl. supervisor mode) što praktično znači da ima vršne ovlasti nad svim sklopovljem računala (npr. može mijenjati sadržaje registara pojedinog uređaja ili samog centralnog procesora, upravljati prekidima, izvršavati privilegirane naredbe na centralnom procesoru, pristupati ovlaštenim adresnim prostorima i sl). Za svaki program koji ima takve ovlasti (odnosno postavljenu takvu zastavicu) se smatra da ne smije nikada doći u nedefinirano odnosno neispravno stanje, budući da time najčešće dovodi do pada cijelog operacijskog sustava i svih korisničkih aplikacija. Ovakva podjela načina rada ima hardversku podlogu, pa većina modernih procesora ima nekoliko načina rada. Specifično, popularni x86 procesori iz PC računala imaju četiri načina rada koji se zovu prstenovi (engl. ring). Jezgra (odnosno aplikacije sa jezgrinim ovlastima) se tipično izvršava u ring0, dok se korisničke aplikacije izvršavaju u ring3.

Nakon paljenja računala i nakon izvršavanja inicijalnog testiranja sklopovlja računala odnosno obavljanja POST (engl. power-on self-test) procedura, potrebno je učitati jezgru sustava. Jezgra nije u stanju sama sebe učitati već to obavlja minijaturni pomoćni program odnosno punilac (engl. boot loader). Njegova je zadaća dohvat jezgre bilo sa čvrstog medija ili mreže, učitavanje u radnu memoriju i predavanje kontrole jezgri. U nekim slučajevima je rečeni punilac toliko rudimentaran da nije u stanju sam učitati jezgru, već predaje kontrolu sekundarnom puniocu (engl. second-stage boot loader) koji je u stanju prepoznati više tipova fizičkih uređaja (mekih diskova, tvrdih diskova, mrežnih uređaja, USB ključeva, CD/DVD uređaja i sl.) i dohvatiti jezgru s njih. Naposljetku se započinje

izvršavati jezgra u nadglednom načinu rada, inicijalizira (postavlja različite varijable na neke pretpostavljene standardne vrijednosti) i pokreće prvi korisnički proces. U većini slučajeva jezgra nakon toga ulazi u "praznu" petlju (engl. idle loop odnosno idle process) u kojem samo reagira na različite vanjske podražaje. Takvi podražaji su npr. prekidi (engl. interrupts) koji potiču od fizičkih uređaja ili pak reakcije na usluge koje zatraže korisničke aplikacije.

Osnovni dijelovi svakog računala bi bili centralni procesor, memorija i ulazno/izlazne jedinice. Upravo se jezgra sustava brine o ispravnoj raspodjeli rečenih resursa među korisničkim aplikacijama, ma koliko ih god bilo. Također je očito da jezgra mora moći omogućiti izvršavanje više aplikacija odnosno procesa pri čemu su također potrebne i usluge sinkronizacije procesa, međusobne komunikacije procesa, dijeljene memorije među procesima i sl. Jezgra se mora pobrinuti da svaka aplikacija dobije vlastiti adresni prostor i da se može izvršavati konkurentno sa ostalim aplikacijama. Moderni operacijski sustavi su u stanju izvršavati aplikacije istovremeno zbog više fizičkih procesora, ali i prividno istovremeno kad je broj aktivnih aplikacija veći od broja procesora. Takva jezgra je višezadaćna (engl. multitasking), a uglavnom funkcionira na takav način da svaka aplikacija odnosno proces dobiva određeni vremenski odsječak u kojem je procesor izvršava, te se zatim se stanje takvog procesa pohranjuje odnosno dešava se izmjena konteksta (engl. context switch) i sljedeći proces dolazi na svoj red. Kakav će biti redoslijed posluživanja procesa određuje politika posluživanja (engl. process scheduling policy), a postoje različiti tipovi posluživanja: kružno, stepeničasto, proporcionalno, težinsko i prioritetno, itd.

Osim sa procesima, jezgra sustava mora upravljati i memorijom i to na siguran način. Procesi za adresni prostor koji koriste najčešće dobivaju virtualne adrese koje mogu i ne moraju biti u stvarnoj fizičkoj memoriji. Pri tome imamo najčešće dva stroga odijeljena prostora: jedan koji je namijenjen korisničkim aplikacijama (engl. user space) i jedan koji je za jezgru i njene potrebe (engl. kernel space). Korisničkim aplikacijama nije nikad dozvoljeno ni čitanje ni pisanje u jezgrin prostor, da ne bi dovele u pitanje sigurnost i pouzdanost sustava. Virtualne adrese također omogućavaju i uniformnu raspodjelu različitih adresnih prostora aplikacijama kao i to da se oni ne preklapaju za pojedine aplikacije. Naravno, tu je također prisutna apstrakcija pa se same aplikacije ne moraju brinuti o kakvoj je memoriji riječ i gdje se ona nalazi. Također smo spomenuli i različite fizičke uređaje o kojima se jezgra mora brinuti. Jezgra u svakom trenutku mora znati stanje uređaja i njegovu fizičku smještenost (sabirnica, adresa, itd). Jezgra najčešće sadrži niz upravljačkih programa (engl. drivers) za pojedine uređaje koji omogućavaju detekciju i inicijalizaciju uređaja, ulazno/izlazne operacije nad istima i sl. Takvi jezgrini programi su u stanju prepoznati neispravan rad uređaja, pojavu novog uređaja na sustavu i ostala razna stanja pojedinog uređaja. Naposljetku, jezgra i aplikacije moraju imati neki zajednički jezik odnosno jezgra mora pružati neko uniformno sučelje prema aplikacijama kako bi aplikacije mogle pozivati i dobivati njene usluge. Takvo sučelje može biti npr. u formi C biblioteke koja s jedne strane pruža sučelje u visokom jeziku a s druge strane interno obavlja jezgrine pozive (engl. system calls).

Što se samog dizajna i naravno same implementacije jezgre tiče, postoji par osnovnih tipova. Dva najinteresantnija i ujedno najviše suprotstavljena tipa su monolitna jezgra (engl. monolithic kernel) i mikrojezgra (engl. microkernel).

Kod monolitne jezgre se cijela jezgra izvršava u jezgrinom prostoru i to upravo u nadglednom načinu rada. Za takvu jezgru je karakteristična vrlo snažna apstrakcija nad sklopovljem kao i sučelje prema aplikacijama sa vrlo bogatim skupom sistemskih poziva i inih primitiva. Unatoč činjenici da su razne funkcionalnosti najčešće implementirane u različitim dijelovima jezgre, stvarno odvajanje takvih dijelova je u praksi nemoguće zbog vrlo visoke razine isprepletenosti samog izvornog koda i podatkovnih struktura. Takva visoka razina međusobne integracije svih dijelova utiče na dobre performanse zbog minimalnih nivoa apstrakcije između samih dijelova kao i visoke efikasnosti koja je posljedica takvog dizajna. Glavna mana ovakvog pristupa je prvenstveno problem da greška u bilo kojem dijelu ovakve jezgre najčešće utiče na pad cijelog sustava, budući da sve komponente nužno vjeruju jedna drugoj. Poznatiji primjeri monolitnih jezgri su: Linux (postojanje modula ne mijenja činjenicu da je riječ o monolitnom dizajnu jezgre), BSD jezgre, Solaris, DOS, većina Unix jezgri kao i

Windows jezgra (hibridnog dizajna budući da je dio jezgrinih modula implementiran u vidu korisničkih aplikacija, no smatra se u osnovi monolitnom jezgrom).

Mikrojezgra je sa druge strane jezgra minimalističkog dizajna: aplikacijama se pružaju tek najosnovnije nužne usluge (odnosno sistemski pozivi), a to su već spomenuto upravljanje adresnim prostorom, vremenskim prekidima, dretvama (engl. threads) pojedinog procesa te komunikacijom među procesima. Sve ostale dodatne usluge poput grafičkih sučelja, mrežnog stoga i sl. su implementirane u korisničkom prostoru u vidu posebnih servisa (engl. servers). Zbog pojave takvih servisa izvan jezgrinog prostora, kod mikrojezgri dodatno dolazi do izražaja potreba za vrlo efikasnom komunikacijom među procesima (engl. inter-process communication) kako je prenošenje poruka jedan od glavnih aspekata rada. Zbog toga što se daleko više poruka prenosi iz jezgrinog prostora u korisnički prostor i obrnuto, mikrojezgre obično imaju nešto lošije performanse od monolitnih jezgri. Spomenuti dodatni servisi su u praksi obične korisničke aplikacije koje se mogu pokretati i gasiti bez loših posljedica za ostatak sustava: jedina njihova privilegiranost je da mogu pisati i čitati neke dijelove memorije koji su nedostupni ostalim procesima. Glavni problem takvog pristupa gdje se neki sistemski servisi mogu gasiti i paliti po potrebi je nestanak strogo definiranog stanja kao kod monolitnih jezgri. Dakle, sasvim je moguće da će u nekom trenutku nestati kakav servis poput mrežnog stoga i da će sve korisničke aplikacije koje su ovisile o mrežnom stogu jednostavno doći u nedefinirano stanje. Iz takvog razloga većina operacijskih sustava koji su bazirani oko mikrojezgri vodi računa o stanju pa omogućava pauziranje aplikacija koje čekaju da se povrati određena usluga i sl. Primjeri mikrojezgri su također prisutni u modernim operacijskim sustavima: Minix, AmigaOS, Mach (GNU Hurd, XNU odnosno Mac OS X), QNX, L4 obitelj, Symbian OS, Singularity.

Linux kernel je jezgra operacijskog sustava napravljena nalik na Unix jezgru. Rečena jezgra je izdana pod GPLv2 (GNU General Public License, inačice 2), te se najčešće može naći u GNU/Linux operacijskom sustavu, unutar različitih Linux distribucija. Linux je isključivo monolitna jezgra koja podržava višezadaćnost, virtualnu radnu memoriju, dijeljene biblioteke, učitavanje ekstenzija (modula) po potrebi, izvršne datoteke koje se učitavaju u dijeljenu memoriju i ravnaju po CoW principu, upravljanje memorijom, rad na višeprocesorskim računalima, višenitnost i implementaciju TCP/IP stoga. Praktički se sva Linux jezgra izvršava u ring0 (jezgrinom) prostoru i s potpunim pristupom svim hardverskim dijelovima računala.

Linux inicijalno nije napisan da bude prenosiv (Linus Torvalds je zamislio da radi samo na 386/486), no tijekom vremena prenesen je na brojne arhitekture: ARM, Alpha, Sparc, IA-64, S/390, IA-32, 68020, Power, SPARC i sl. Trenutno Linux uspješno radi na platformama od ručnih uređaja do velikih mainstream poslužitelja, tzv. big-iron poslužitelja. Linux je napisan u C jeziku i uglavnom (s iznimkom najsvježijih inačica Intel prevoditelja) se može kompilirati jedino sa GNU C prevoditeljem zbog uporabe GCC ekstenzija. Specifičnije rutine su napisane u asembleru u AT&T sintaksi.

Što se pak same povijesti razvoja tiče, prva inačica 0.01 je izdana krajem 1991. godine. Tijekom razvoja možemo izdvojiti ove veće periode:

Danas možemo jezgru kernela prepoznati kao tri ili četiri broja međusobno odvojena točkama. Dakle opća forma inačice je A.B.C[.D], pri čemu uglate zagrade označuju da je D opcionalan:

U načinu označavanja Linux 2.6 jezgre postoji još varijacija, gdje se dodaju pojedine dvoslovne oznake koje prvenstveno označavaju nova "stabla" u razvoju i njihove razvijatelje. Takva stabla donose nove i još nedovoljno testirane zakrpe, raznorazne specijalizirane dodatke i sl. Istraživanje i korištenje takvih jezgri nije preporučljivo u produkcijskim okolinama.

**Općenito o CARNet kernel paketu**

Kao što već znate, CARNetovi poslužitelji obično koriste unaprijed izgrađeni kernel-2.6-cn paket. On se temelji na stabilnoj Linux 2.6 jezgri (dostupna sa adrese http://www.kernel.org/) , razvojnim Grsecurity dodatcima (http://www.grsecurity.net/~spender/) te L7-filter dodacima (http://sourceforge.net/projects/l7-filter/files/). Stabilnom jezgrom smatramo onu major verziju koja je

već otprije izdana te na kojoj se ne odvija intenzivni razvoj, ali koja ima pokrpane sve poznate probleme. Na primjer, trenutna razvojna verzija Linux jezgre je 2.6.24-rc4, a trenutna stabilna inačica je 2.6.23.9; dakle u 2.6.23 inačici je već 9 podverzija koje imaju ispravljene raznorazne greške, dok se znatnije promjene najranije očekuju u 2.6.24 inačici.

Ukratko navedimo što pojedini dodatak donosi:

**Raspored CARNet kernel paketa**

Sam izgled kernel-2.6-cn paketa možemo ukratko raščlaniti na:

**Podržani hardver u CARNet kernel paketu**

**Instalacija CARNet kernel paketa**

Do recentne i stabilne Linux jezgre najjednostavnije možete doći sa instalacijom CARNet Linux kernel paketa:

Prilikom same instalacije paketa, automatski se generira LILO konfiguracija (datoteka /etc/lilo.conf) i postavlja preporučena jezgrina konfiguracija (/etc/sysctl.conf). Takodjer, paket kreira novu grupu "proc" sa GID-om 99. Osim root grupe, jedino rečena proc grupa ima privilegije čitanja potpunog /proc stabla; stoga se prakticira u tu grupu staviti specifične servise koji trebaju pristup /proc stablu - to su snmpd, oidentd i sl.

**Vlastita kompilacija jezgre**

Linux jezgra se relativno lako i bezbolno rekompilira. Nažalost, najveći problem je sama konfiguracija odnosno rekonfiguracija, ako koristite već pripremljenu config datoteku iz kernel-2.6-cn paketa. Tijekom vremena kako se razvijao računalni hardver i softver je vrlo narasla količina opcija koje je moguće podesiti (omogućiti ili onemogućiti). Shodno tome, prilikom rekonfiguracije potrebno je proći desetine menija i stotine opcija što je donekle zamoran posao. No još gore, očekuje se prilično dobro znanje o pojedinoj opciji i samom hardveru kojeg imate. Prije ikakvog postupka vlastite rekompilacije preporučamo pročitati par relevantnih dokumenata, poput http://www.faqs.org/docs/Linux-HOWTO/Kernel-HOWTO.html te http://www.digitalhermit.com/linux/Kernel-Build-HOWTO.html.

Da bi uopće mogli kompilirati jezgru, potrebno vam je minimalno C razvojno okruženje. Ono je dobavljivo na sljedeći način:

Najjednostavnije je početi od već poznate i ispravne konfiguracije poput one iz kernel-2.6-cn paketa. Samu jezgru možete preuzeti bilo instalacijom kernel-2.6-source-cn paketa (pa će se pojaviti u /usr/src/linux-2.6 direktoriju) bilo direktnim skidanjem sa izvornog mjesta, a to je https://www.kernel.org/. U prvom slučaju dobivate sve dodatke integrirane s jezgrom, a u drugom slučaju potrebno je to ručno odraditi. Dakle, pokažimo prvi način - instalaciju već priređenog izvornog koda za CARNet kernel-2.6-cn paket:

Druga varijanta je tek neznatno kompliciranija: prvo ćemo dohvatiti svježu jezgru (koja je aktualna uvijek možete provjeriti na https://www.kernel.org/):

Dohvatimo i posljednji Grsecurity dodatak, uvijek dostupan u razvojnom direktoriju samog autora (http://www.grsecurity.net/~spender/). Pazite, verzija u Grsecurity dodatku mora odgovarati verziji jezgre koju ste skinuli:

Izvorni kod je spreman za rekompilaciju, sad možemo iskoristiti postojeću konfiguraciju iz kernel-2.6-cn paketa:

Nakon završenog konfiguriranja (postupak nadalje vrijedi za obje varijante pribavljanja izvornog koda), potrebno je prevesti jezgru i module:

Te instalirati na pravo mjesto:

Jasno, prije kompilacije je moguće skinuti i po volji dodati raznorazne druge dodatke, što se najčešće obavlja naredbom patch.