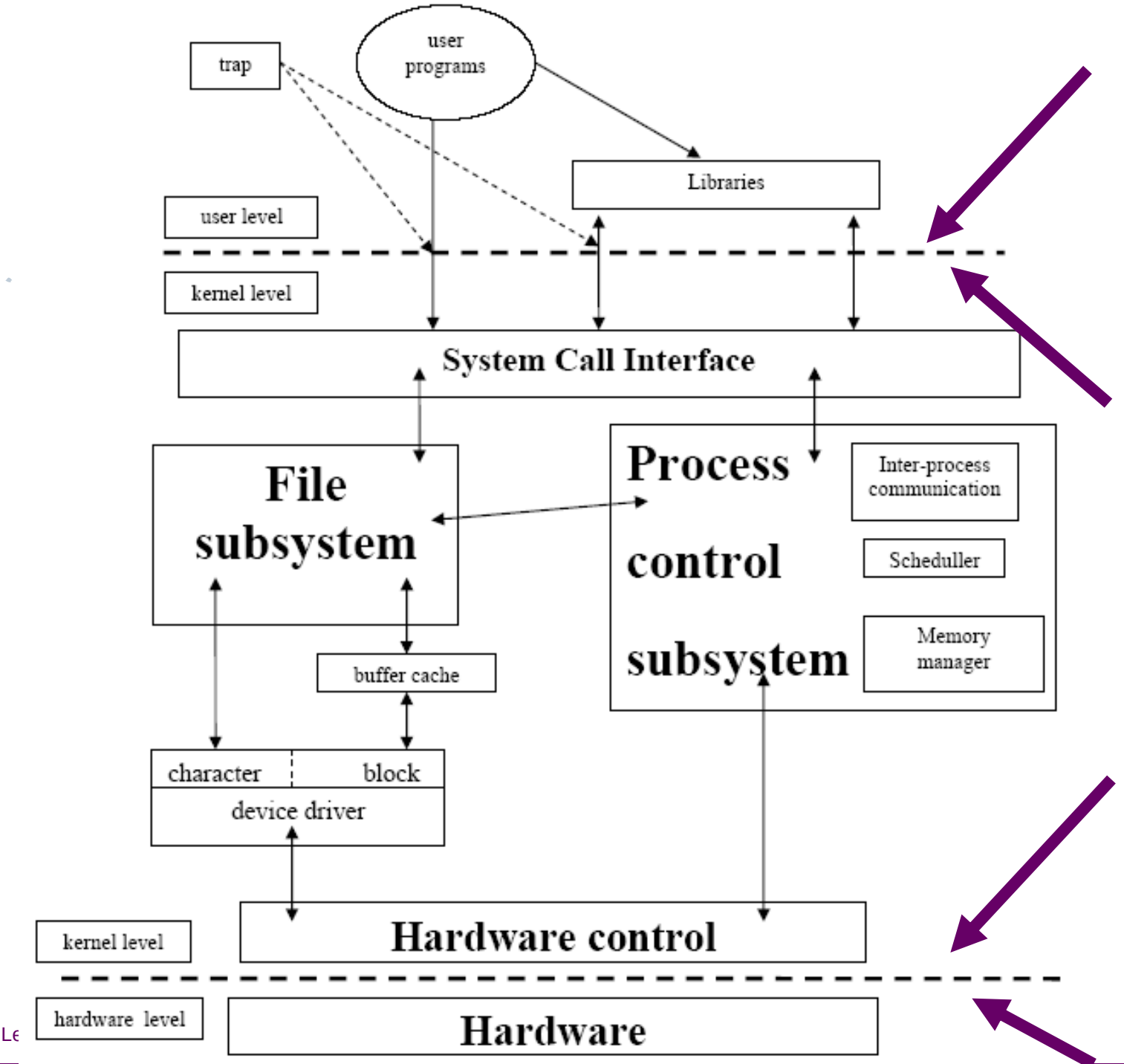
Uvod u Kernel

- **Arhitektura UNIX operativnog sistema**
  - ☞ **user level**
  - ☞ **kernel level**
  - ☞ **hardware level**


trap
user programs
Libraries
user level
kernel level
System Call Interface
File subsystem
Process control subsystem
Inter-process communication
Scheduller
Memory manager
buffer cache
character
block
device driver
kernel level
Hardware control
hardware level
Hardware

Arhitektura UNIX operativnog sistema

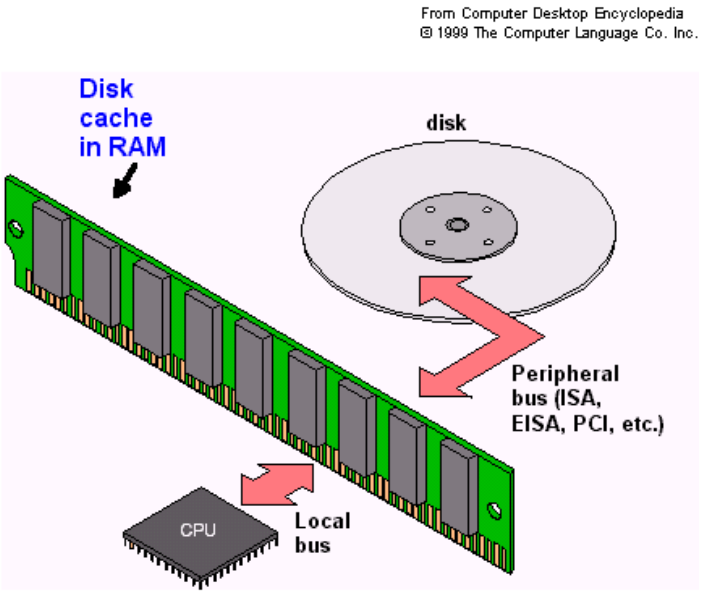
- **Dva entiteta**, **datoteke** i **procesi** su
- **2 centralna koncepta** u **UNIX sistem modelu.**
- **Imamo 3 nivoa arhitekture**:
  - **korisnički nivo**
  - **kernelski nivo**
  - **hardverski nivo**
- **SC intefejs** i **library interfejs** predstavljaju **granicu**
- između **korisničkog programa i kernela.**
- **SC liče na obične pozive funkcije u C programu**,
  - a **biblioteke mapiraju ove funkcijske pozive**
  - **u primitive neophodne da se uđe u OS.**
- **Asemblerski programi** mogu pozvati **SC direktno**
  - **bez poziva sistemskih biblioteka**.
- **Programi često koriste pozive za sistemske biblioteke**
- **koji se linkuju sa programom**
  - in compile time
  - in execution time

System Calls for FS

- **FS (File Subsystem)**
  - **upravlja datotekama**
  - **alocira prostor** za **datoteke**
  - **administrira slobodan prostor**
  - **kontroliše pristup** datotekama
  - **obezbeđuje podatke iz datoteka** korisnicima.
- **Interakcija procesa** sa **FS** se odvija **preko skupa sledećih SC**:
  - **open (open file for reading and writing)**
  - **close**
  - **read**
  - **write**
  - **lseek**
  - **stat (querry the attribute of file)**
  - **chmod**
  - **chown**

buffered v raw access

- **FS pristupa podacima** na **2 načina**:
- **1. buffered access**
  - **preko cache bafera** (**sve ide kroz kes**)
  - **FS ima specifični interakciju**
    - **sa I/O block uređajima kroz keš,**
  - **keš bafer reguliše protok podataka**
    - **između kernela i I/O uređaja**.



- **2. raw access**
  - **bez cache bafera**
  - **FS pristupa blok I/O uređajima direktno, bez keš bafera**
  - ovaj pristup se naziva **raw**.
  - na isti način se upravlja svim uređajima **koji nisu blok orijentisani**.

- **Device drivers** su **kernelski moduli** koji **upravljaju radom I/O uređaja**.

System Calls za procese

- **Proces control subsystem(PCS)** je **odgovoran za**
  - **sinhronizaciju procesa**
  - **IPC**
  - **za upravljanje memorijom**
  - **za CPU scheduling.**

- **PCS** i **FS komuniciraju**
  - **kada se puni program iz FS u memoriju radi izvršenja,**
  - kada **PCS čita executable file** u **memoriju** pre nego što je **izvrši**.

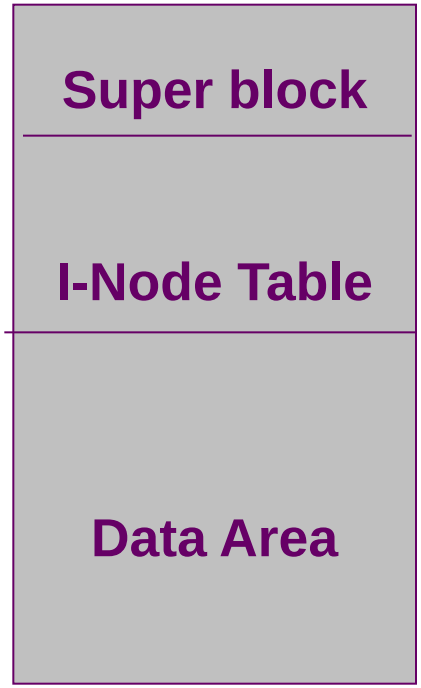
- **SC** iz **PCS su**:
- **fork** (**create** a **new process**)
- **exec** (**overlay** the image of a program onto running process)
- **exit** (**finishing** executing process)
- **wait** (**synchronize process execution with** the **exit** of a **previosly forked process**)
- **brk** (**control** the **size of memory** allocated to a process)
- **signal** (**control** process **response** to **extraordinary events**)

Part of PCS

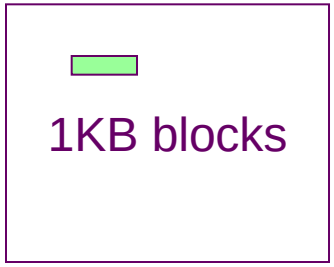
- **1. Memory Management module**
  - kontroliše **alokaciju memorije**.
  - Tu svakako spadaju i **dodatne funkcije**
  - vezane za **virtuelnu menoriju**, kao što su **swaping** i **DP**.
- **2. CPU Scheduler: alocira CPU procesima**:
  - **nastupa posle blokiranja procesa ili**
  - **posle isteka time quantuma**
- **3. IPC**: postoji **više formi** IPCa: počevši od
  - **asinhronog signaliziranja događaja**,
  - do **sinhronih prenosa poruka** između njih
- **4. Hardverska kontrola:** je **odgovorna**
  - za **upravljanje prekidima** i
  - za komunikaciju sa hardverom-mašinom.
- **Prekidni programi** se **ne servisiraju kao specijalni procesi**
  - već kao **specijalne funkcije u kernelu**
  - a **u kontekstu procesa koji se izvršavaju**.

UNIX FS

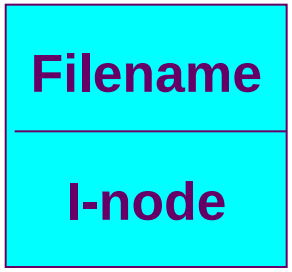
- FS Layout



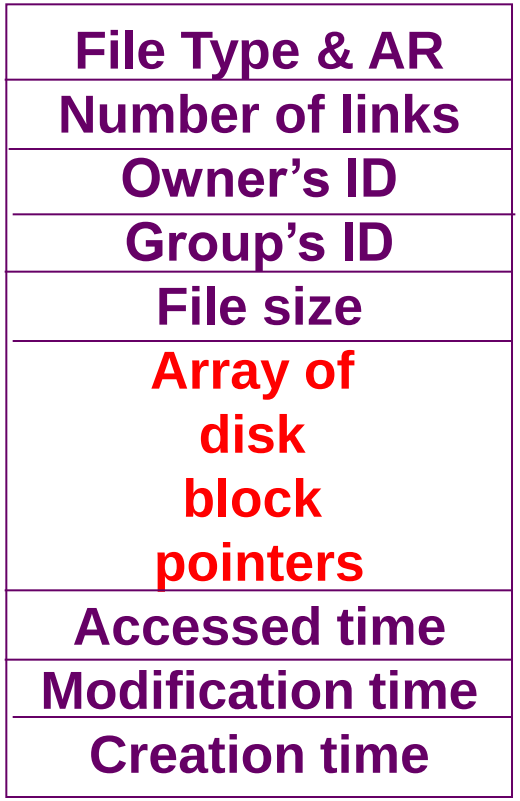
DATA area



File-info

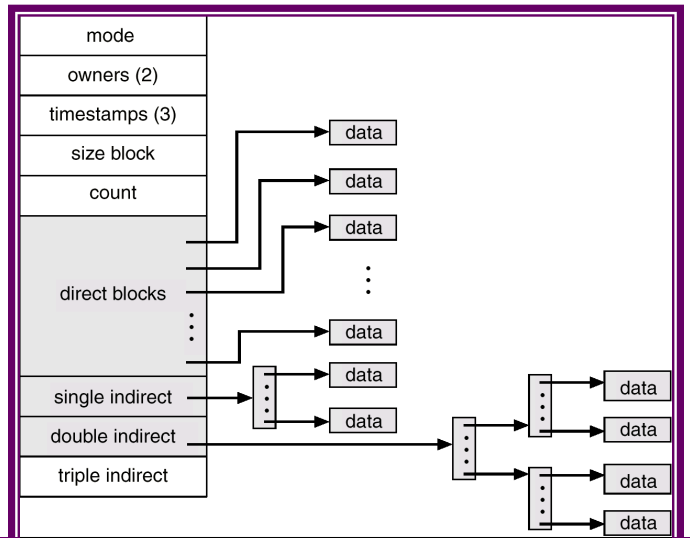


I-node



Opšti pregled FS

- **Interna repreznetacija datoteke** je definisana **preko inode strukture** koja opisuje
  - **layout datoteke** na disku
  - **druge informacije** kao što je vlasništvo, prava pristupa i vremena pristupa.
- **Svaka datoteka** ima **jedan unikatni inode**,
  - ali **može imati više imena** (**hard links**).
- **Kada proces traži** neku **datoteku** po **imenu**,
  - kernel analizira **svaku komponentu** u **pathname**,
  - proverava **da li proces ima prava da pretražuje** u **toj grani**,
  - a ako dođe do **poslednje grane** otvara **traženi inode**.
- Kada proces kreira **novu datoteku**,
  - kernel **mora** da **dodeli novi,**
  - **slobodni inode.**
- **Inodes** se **čuvaju** u **inode tabelu** na **disku**,
- ali radi **ubrzanja** rada
  - **kernel otvorene inodes**
  - **čuva** u **memorijskoj (in-core) inode tabeli**.

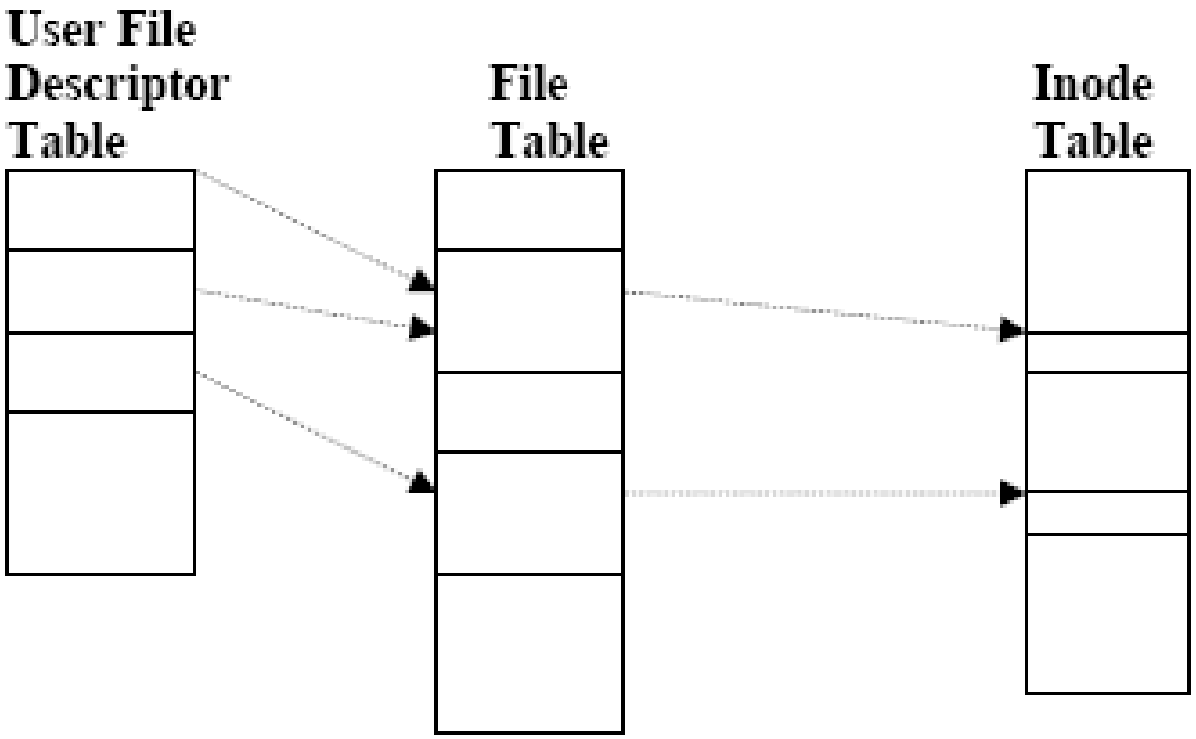


FT, UFDT

- **Kernel sadrži još 2 dotatne strukture podataka,**
- **FT=file table**
  - FT je **globalna kernelska struktura**
- **UFDT = user file descrpitor table**.
  - **UFDT je tabela** koja se dodeljuje **svakom procesu**.
- **Kada proces otvara** ili **kreira novu datoteku**,
  - **kernel alocira** po **jedan ulaz**
  - iz **svake** od **ove 2 tabele**.
- **Stanje "live" datoteke** i **korisničkog pristupa** toj datoteci je dato preko **3 tabele**:
- **1. inode table**
- **2. FT**: **čuva zapis** o **offsetu u datoteci**
  - **gde će se sledeći upis ili čitanje startovati** i
  - **prava pristupa koje ima openning proces**
- **3. UFDT**: **identifikuje sve otvorene datoteke** za **taj proces**

FT, UFDT, inode table

- Kada se **otvori ili keira datoteka**, kernel vraća deskriptor koji je index u UFTD.
- Kada se potom obavlja **read ili write**,
  - na bazi deskriptora se ulazi u UFDT,
  - iz koga se čita ulaz u FT, a
  - onda se preko inode tabele realizuje odgovarajući read ili write.



Disk layout

- Disk može biti izdeljen **na više sistema datoteka** od kojih ima svoj logički broj. Konverzija između **logičke adrese <FS, logical_number>** i **fizičke adrese <cyl, head, sector>** je zadatak **disk drajvera**.
- Diskovi se sastoje of fizičkih blokova = 512. FS se sastoji od logičkih **sistemskih blokova** veličine 1K, 2K, 4K, 8K.
- **FS se sastoji od više komponenti**:

boot block	super block	inode list	data blocks

- **Boot block** okupira početak FS-a, obično je to prvi sektor, i sadrži boot code za inicijalno podizanje operativnog sistema.
- **Super block** opisuje stanje FS-a,
  - **koliko je veliki**,
  - koliko **maksimalno datoteka** može sadržavati,
  - **gde se nalazi informacija o slobodnom prostoru u FS** i razne druge informacije.
- **Inode list** je tabela koja sledi iza superbloka, administratori specificiraju veličinu ove tabele kada konfigurišu FS.
  - **Kernel pronalazi inodes indeksiranjem u inode tabelu.**
  - **Specijalan inode je root inode za taj FS:**
  - **to je onaj inode preko kojeg je direktorijumska struktura tog FS**
  - **raspoloživa nakon uspešne mount komande**
- **data area**: počinje odmah iza inode tabele i sadrži datoteke i direktorijume

Processes

- Proces je program u izvršavanju i sastoji se od **3 funkcionalne celine**
  - **text**
  - **data**
  - **stack**

- U princupu **nema preklapanja** ovih delova sa drugim procesima a sva komunikacija između procesa se odvija preko IPC-a. **Postoje izuzeci** od ovog kao što su deljivi kod segmenti (**shared code segment**) i deljiva memorija (**shared memory**).

- **Praktično,** proces na **UNIX sistemu** je celina kreirana preko **fork SC**. Svaki proces izuzev procesa 0 se kreira kada neki drugi proces izvrši fork sistemski poziv, pri čemu se taj proces naziva roditelj, a kreirani proces se naziva dete. Svako dete ima samo jednog roditelja, a jedan roditelj može imati puno dece. Kernel svakom procesu dodeljuje **jedinstveni broj PID**.

- **Proces sa PID=0** je specijalan proces koji se kreira ručno kada se UNIX podiže, a potom taj proces forkuje svoje prvo dete koje postaje **proces init** sa **PID=1**, dok proces sa PID=0 postaje **swapper** process. **Proces init** je **predak svih procesa na UNIX-u** i **svi ostali procesi imaju specijalnu vezu sa njim**.

Struktura izvrsnog programa

- **Izvršni prgoram** se sastoji od **više funkcionalnih delova**:
  - skup zaglavlja „header“ koji opisuju atribute datoteke
  - text = programski kod
  - data sekciju inicijalizovanu (varijable koje imaju početnu vrednost kada se program startuje)
  - data sekciju koja se ne inicijalizuje bss
  - razne sekcije kao što je simbolička tabela na primer

- Kada se program puni u memoriju preko exec SC,
- **minimalno 3 memorijska regiona** moraju mu se dodeliti:
  - **text**
  - **data**
  - **stack**

- Text i data sekcija odgovaraju text i databss samog programa

Struktura izvrsnog programa-stack

- **stack automatski kreira** i njegova veličinu kernel automatski prilagođava u **runtime**.

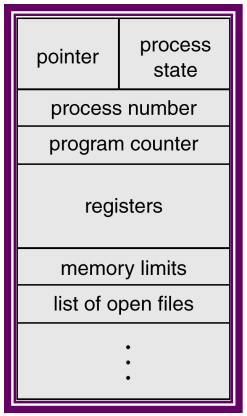
- Stack se sastoji od **logičkih stack frame-ova, koji se**
  - **guraju na stek** (**push)** kada se **funkcija pozove** i
  - **skidaju sa steka** (**pop**) kada se obavlja **povratak iz funkcije**.

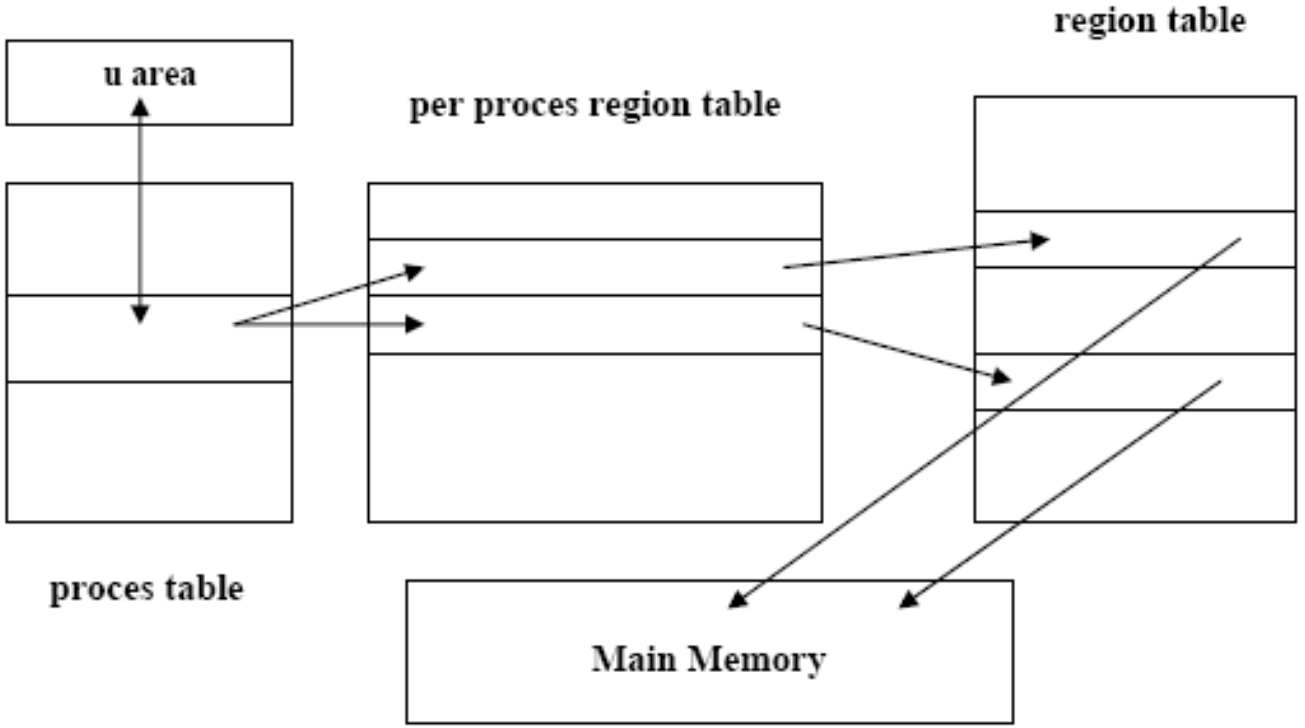
- **Stack frame** sadrži
  - **adresu samog frame**,
  - **povratnu adresu funkcije**,
  - **parametre funkcije**,
  - **njene lokalne varijable**,
  - **podatke potrebne za regeneraciju prethodnog stek frame** uključujući PC i SP u vreme funkcijskog poziva.

- Kako proces na UNIXu može da se izvršava u 2 moda, kernelski i korisnički, za svaki mod mora da se koristi poseban stek.
  - **Kernelski stek** se sastoji od stek frames za funkcije koje se izvršavaju u kernelskom modu i sve je isto kao u korisničkom steku po strukturi .
  - Ukoliko nema sistemskih poziva, trapova i prekida, kernelski stek za proces = 0..

PT, u-area

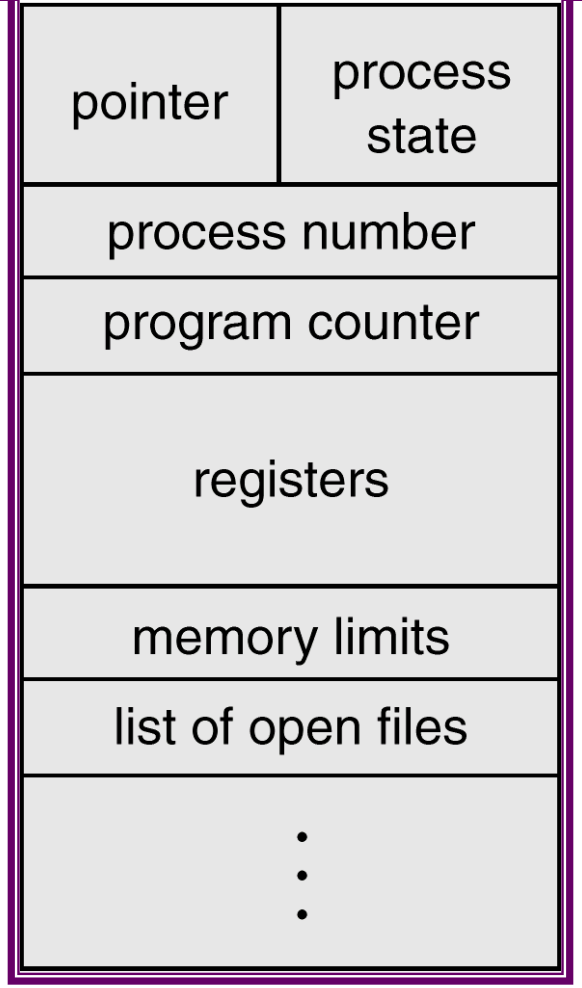
- Svaki proces ima **2 ulaza**:
  - **ulaz u kernelskoj tabeli procesa PT**
  - **ulaz u u-area (u = user)** a to je područje kojim manipuliše isključivo kernel
- **PT ulazi sadrže odnosno** ukazuju na **per proces region table PPRT**, čiji **ulazi ukazuju** na **region tabelu**. **Region je kontunalni adresni prostor** za jedan proces, kao što je text, data ili stek. **Region tabela (RT),** u svojim ulazima **opisuje atribute regiona** (text-data, private-sharable) i **lokaciju regiona** u memoriji. **Upotreba RT i PPRT omogućava** efikasno **deljenje regiona**.





PT entry

- Objasnimo vezu između ovih tabela.
  - **PT ukazuje na PPRT,**
  - **PPRT ima ukazivače na glavnu RT**
  - **RT opisuje regione**
  - **Ulaz u PT i ulaz u u-area**
  - sadrže **kontrolne i statusne** informacije
  - o **svakom procesu.**



- **Polja u PT ulazu su:**
  - **polje stanja** (**state fileld**)
  - **UID:** identifikatore koji opisuju koji user je **vlasnik procesa**
    - (user ID odnosno UID)
  - **skup opisivača događaja** kada se proces **suspenduje**
    - (in the sleep state)

u-area entry

- Ulaz u u-area sadrži informacije o procesu koje su potrebne jedino kada se proces **izvršava**:

- **u-area:**
- **polja u njemu su:**
  - **ukazivač na PT ulaz** za proces koji se **trenutno izvršava**
  - **parametre tekućeg SC, povratne vrednosti i error kodove**
  - **file deskriptore za sve otvorene datoteke**
  - **tekući direktorijum i tekući root**
  - **limiti za veličinu procesa i veličinu datoteka**

- **Kernel uvek direktno pristupa poljima u u-area samo** za **proces** koji **se tekuće izvršava**

System calls and regions

- Po pitanju SC i regiona važe sledeća pravila:
- **fork SC:**
  - **kernel duplicira adresni prostor** od procesa roditelja,
  - **dozvoljavajući da oba procesa dele regione uvek kad je moguće**
  - i **praveći kopije kada se to mora**

- **exec SC:**
  - **kernel alocira regione za text, data i stek,**
  - a **prethodno oslobođa sve regione** koje je **proces imao pre exec SC**

- **exit SC:**
  - **kernel oslobođa sve regione** koje je proces posedovao

Kontext procesa

- Kontekst procesa je njegovo stanje koje se definiše:
- **CPU context**
  - vrednost registara CPU koje proces koristi
- **memory context**
  - njegov text
  - vrednost globalnih promenljivih i data struktura
  - sadržina korisničkog i kernelskog steka
- **IO context**
  - all open files
- **Kernel structures**
  - vrednosti ulaza i u PT i ulaza iz u-area

Kontext procesa i CSw

- OS izvršava proces, preciznije **OS izvršava kontekst procesa**.
- **Kontekst procesa se čuva u 3 situacije:**
- **1. CSw**
  - Kada kernel **odluči da izvršava neki drugi proces**, mora se obaviti kontext switch, **CSw**.
  - Kernel dozvoljava prebacivanje konteksta samo **pod određenim uslovima**.
  - Kada obavlja **CSw**, kernel mora da **sačuva dovoljno informacija** o procesu koji se suspenduje kako bi kasnije mogao da ga nastavi.
- **2. user->kernel mod**
  - prilikom prebacivanja iz user moda u kernelski mod, kernel **mora sačuvati dovljno informacija** kako bi proces nastavio izvršavanje tamo gde je stao, nakon povratka u user mod.
  - Naravno u ovom slučaju se ne menja kontekst procesa već samo mod.
- **3. Interrupt**
  - Kernel opslužuje prekide u kontekstu jednog istog procesa koji se prekida, pa se nastavi.
  - Opsluživanje prekida se **ne realizuje preko novih procesa**, ali se radi uvek u kernelskom modu.
  - Prekinuti proces **može biti i u user modu i u kernelskom modu**, a prilikom prekida mora se sačuvati dovoljno informacija da prekinuti proces može da se nastavi.

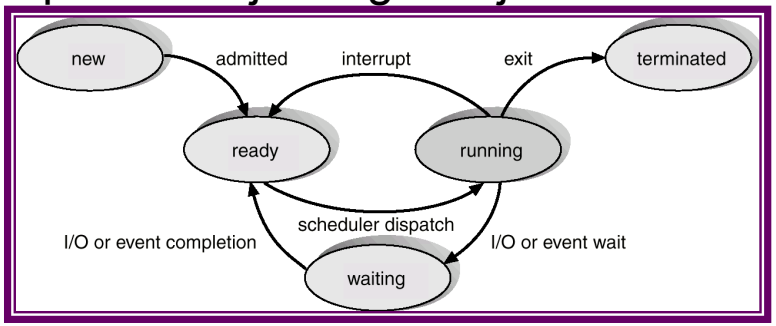
Stanja procesa

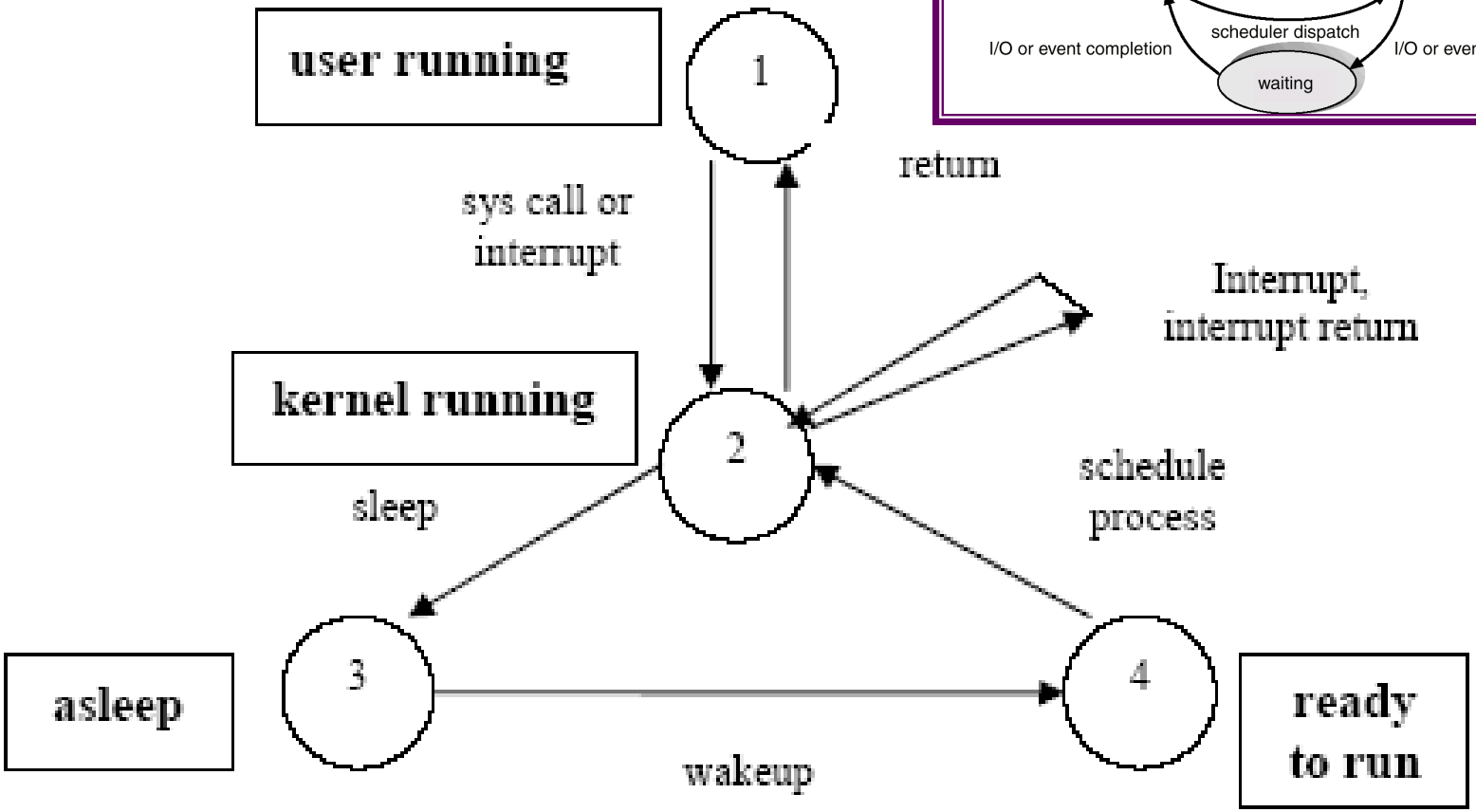
- Proces se može naći u nekoliko stanja:
- (**running in user mode**)
  - Proces se trenutno izvršava u korisničkom modu
- (**running in kernel mode**)
  - Proces se trenutno izvršava u kernelskom modu
- (**ready**)
  - Proces je ready: ne izvršava se, nego čeka da scheduler prozove
- (**asleep in memory**)
  - Proces je uspavan: proces blokira svoje izvršavanje zato što ne može dalje da nastavi čekajući na nešto, na primer na I/O da se završi

- Zato što procesor može izvršava samo jedan proces u vremenu, samo jedan proces može biti u stanju 1 ili 2. Svi ostali procesi, osim CPU aktivnog, su u stanju 3 ili 4.

Tranzicije procesa

- Procesi često menjaju svoja stanja po dobro poznatim pravilima, koja su prikazana na sledećoj slici, gde krug predstavlja stanje, a ivica sa strelicom predstavlja događaj koji izaziva da se proces pomeri iz jednog stanja u drugo.





preemptions

- Više procesa mogu biti u memoriji, a takođe više njih mogu raditi u kernelskom modu. Da ne bi došlo do narušavanja kernelskih struktura podataka, **kernel ne dozvoljava bilo kakvo prebacivanje konteksta** a **takođe kontroliše i prekide**.

- **Kernel dozvoljava prebacivanje konteksta** samo kada se proces prebacuje iz stanja **kernel running** u stanje **asleep in memory**.
- **Proces koji radi u kernelskom modu ne može biti preemptovan** od bilo kog drugog procesa, tako da se za kernel kaže da je **non-preemptive** i na taj način se rešava problem ME u kernelskom modu (samo jedan proces izvršava CS u kerneskom modu i ne može biti preempteed).
- **Čak i prekidi mogu biti suspendovani** ako mogu da dovedu do problema inkozistencije podataka u kernelu. Takav deo koda se naziva **CriticalSection** u kernelu i tada kernel podiže CPU nivo da su većina prekida blokirana, samo dok je proces u CS (naravno u kernel modu).

- **UNIX kernel sebe štiti** tako što **dozvolava CSwitch samo na jednom mestu** i **što blokira prekide koji su opasni u svojim CS** (na primer, dok ažurira buffer queue pointere, blokiraće disk prekidi).

Sleep and wakeup

- Proces koji radi u **kernelskom modu ima veliku autonomiju** da odluči kako će da reaguje na sistemske događaje. Ako već mora da čeka na nešto, poželjno je da se **uspava-blokira, ali to je odluka samog procesa**.
- Na drugoj strani, **interrupt handler ne sme da se uspavljuje**, jer ako to učini, prekinuti proces bi bio uspavan po defaultu.

- Procesi se uspavljuju zato što čekaju da se neki događaj desi: završetak I/O operacije, čekanje na drugi proces da se završi, čekanje da resurs postane raspoloživ. **Procesi se blokiraju na događaj**, i kada se on desi prelaze u **ready to run**. Mnogi procesi mogu biti uspavani na isti događaj, a kada se on **desi svi se bude i prelaze u ready queue, nema neposrednog izvršavanja**.

- Na primer, proces koji se izvršava u kernelskom modu može **lock-ovati neke data strukture pre nego što ode na spavanje**. **Svi drugi moraju da čekaju na unlock, a dotle i oni idu na spavanje**.

Lock example

- Kernel **implementira locks** na sledeći način:
- **lock**
  - while(condition is **true**)
    - sleep (event: the condition becomes **false**)
  - set condition **true**
- Proces obavlja **unlock** i budi sve procese koji to čekaju:
- **unlock**
  - set condition **false**;
  - wakeup(event: condition is **false**)

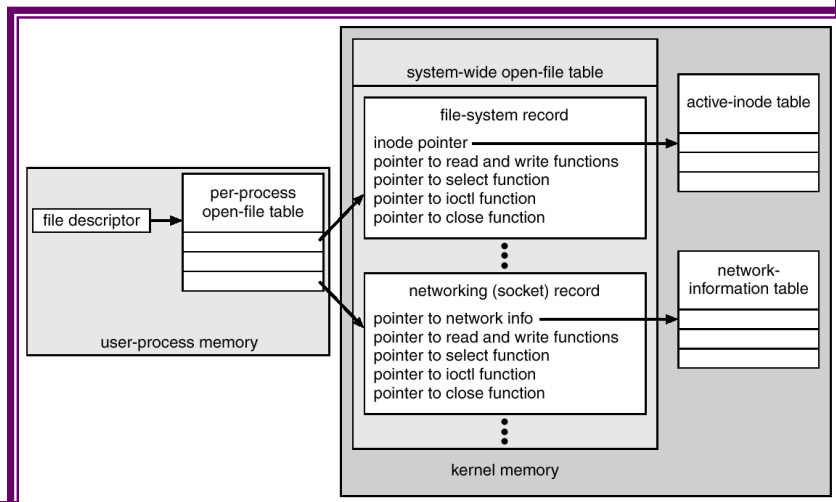
Strukture podataka kernela

- Većina struktura podataka kernela zauzima
  - **tabele fiksnih veličina**
  - **radije**
  - nego dinamički alocirani prostor

- To ima prednosti jer **uprošćava kernelski kod**,
- **ali limitira broj ulaza** u **kernelskim strukturama**.

- Ako se prekorači broj ulaza, korisnik mora da čeka i šalje mu se **poruka o grešci**, za njegov proces nema ulaza u kernelskoj strukturi.

- Postoje i kerneli koji se mogu adaptirati, ali to se kosi sa efikasnošću i jednostavnošću kernela.



Sistemska administracija

- Postoje razni **adminstativni procesi** koji obavlja **razne funkcije**,
  - formatiranje diska,
  - kreiranje FS,
  - podešavanje parametara kernela.

- Za takve procese,
- kernel zahteva **root privileguju**,
- odnosno da takve procese izvodi specijalni account sa atributima superuser-a.

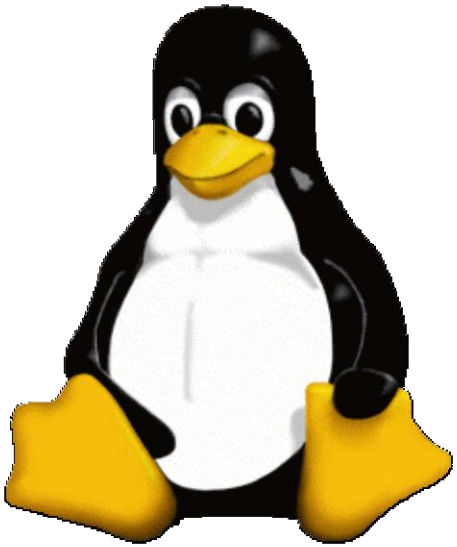
Linux operativni sistem

- **Linux = Linus + Unix**
- **Linux = Unix made by Linus Torvalds**

- Jedna od poslednjih varijanti UNIX operativnih sistema,
- razvoj započeo Linus Torvalds
- **1991. godine na Unverzitetu u Helsinkiju**,
- Torvalds je svoj operativni sistem koji objedinjuje oba standarda, SRV4 i BSD, objavio na Internetu i podsticao druge programere širom sveta da se priključe njegovom daljem razvoju.
- Ubrzo, Linux je postao veoma popularan među računarskim entuzijastima, koji su tražili alternativno rešenje za postojeće operativne sisteme za PC računare (DOS, Windows).
- Linux je svojom koncepcijom, stabilnog a jeftinog operativnog sistema doživeo veliku ekspanziju i popularnost.

Simbol Linux-a i rasprostranjenost

- Simbol Linux sistema je mali pingvin (Tux), prikakazan na slici



GNU/Linux i Open Source Software

- Linux je raspoloživ kao besplatan operativni sistem
- pod GNU GPL licencom (GNU General Public License),
- što važi i za neke druge vrste UNIX sistema,
- kao što su FreeBSD i NetBSD.

- Linux je softver sa otvorenim izvornim kodom (Open Source), što znači da je mu je izvorni kod javno raspoloživ i može biti modifikovan tako da odgovara specifičnim potrebama.

- Linux se može slobodno distribuirati među korisnicima.

- Ovakav koncept je potpuno suprotan konceptu komercijalnog softvera, gde izvorni kod nije dostupan i svaki korisnik mora da plati licencu za korišćenje. Komercijalni softver je baziran na autorskim pravima (copyright laws), koja preciziraju limite koje korisnici softvera imaju u odnosu na izvorni kod, korišćenje i dalje distribuiranje softvera.

- Linux se besplatno može preuzeti sa različitih web-sajtova.
- **GNU stands for “GNU's Not Unix”**

Linux distribucije - definicija

- Brojne profit-orijentisane i neprofitne organizacije čine Linux raspoloživim u formi distribucija, odnosno različitih kombinacija kernela, sistemskog softvera i korisničkih aplikacija. Većina distribicuja sadrži kolekciju CD/DVD medijuma na kojima se nalazi operativni sistem, izvorni kod, detaljna dokumentacija, kao i štampana uputstva za instalaciju i upotrebu sistema. Cene ovakvih distribucija su u većini slučajeva simbolične, osim ako se u distribuciji nalazi komercijalan softver ili je distribucija specifične namene.
- **Osnovna komponenta** svake **Linux distribucije** je **kernel** operativnog sistema.
- Osim kernela i sistemskog softvera, u distribuciji se nalaze:
  - instalacioni alati, softver za podizanje operatvnog sistema (boot loader)
  - razne korisničke aplikacije (kacelarijski paketi - office suite
  - softver za manipulaciju bit-mapiranih slika)
  - serverski paketi
- Većina distribucija je poput Windows sistema, grafički orijentisana prema korisniku, dok su neke distribucije namenjene za sistemske administratore i programere familijarne sa tradicionalnim UNIX okruženjem.

Linux distribucije - pregled

- U poznatije Linux distribucije spadaju:
  - Debian GNU/Linux (http://www.debian.org),
  - Linux Mandrake (http://linux-mandrake.com/en),
  - Red Hat Linux (http://www.redhat.com),
  - Slackware Linux (http://www.slackware.com) i
  - SuSE Linux (http://www.suse.com).
  - Ubuntu Linux (http://www.ubuntu.com)

Opšti pregled Linux sistema

- Linux je višekorisnički,
  - višeprocesni operativni sistem
  - sa potpunim skupom UNIX kompatibilnih alata,
  - projektovan tako da poštuje relevantne POSIX standarde.
- Linux sistemi podržavaju tradicionalnu UNIX semantiku i potpuno implementiraju standardni UNIX mrežni model.
- Linux operativni sistem sastoji se od:
  - kernela
  - sistemskog softvera
  - korisničkih aplikacija
  - programskih prevodioca i njihovih odgovarajućih biblioteka (GCC - GNU C Compiler i C biblioteka za Linux) i
  - dokumentacije
- Sadržaj konkretne Linux distribucije definisan je sadržajem instalacionih medijuma, koji u slučaju nekih Linux sistema uključuju razne FTP sajtove širom sveta.

Linux kerneli - uvod

- Kernel je srce operativnog sistema - on omogućava konkurentno izvršavanje procesa, dodeljuje im memoriju i druge resurse i obezbeđuje mehanizam za ostvarivanje servisa operativnog sistema.
- Kernel štiti korisničke procese od direktnog pristupa hardveru - procesi pristupaju hardveru korišćenjem sistemskih poziva kernela, čime se obezbeđuje jedna vrsta zaštite između samih korisnika.
- Sistemski programi koriste kernel u cilju implemetacije različitih servisa operativnog sistema.

- Svi programi, uključujući i sistemske, funkcionišu na nivou iznad kernela, što se naziva korisnički režim rada, dok se sistemske aktivnosti poput pristupa hardveru obavljaju na nivou kernela, odnosno u kernelskom režimu rada.

- Razlika između sistemskih i aplikativnih programa je u njihovoj nameni: aplikacije su namenjene za razne korisne aktivnosti (kao što su obrada teksta i slike), dok su sistemski programi namenjeni za rad sa sistemom i administraciju.
- Na primer tekst procesor je korisnička aplikacija, dok je komanda mount sistemski program. Razlike između korisničkih i sistemskih programa su ponekad veoma male i značajne samo za stroge kategorizacije softvera.

Linux kernel-verzije

- Tri osnovne verzije Linux kernela su početna verzija, verzija 1.x i verzija 2.x.
- **Početna verzija 0.01**, koju je 1991. godine kreirao Linus Torvalds, podržavala je samo Intel 80386 kompatibilne procesore, mali broj hardverskih uređaja i Minix sistem datoteka. Mrežni servisi nisu imali kernelsku podršku.

- **Verzija 1.0,** nastala u martu 1994. godine, uključivala je podršku za standardne TCP/IP mrežne protokole, BSD-kompatibilni socket interfejs za mrežno programiranje i drajversku podršku za mrežne kartice. Ova verzija je dodatno podržavala ext i ext2 sisteme datoteka, široku klasu SCSI disk kontrolera, kao i brojne hardverske uređaje. **Verzija 1.2 (mart 1995)** je poslednja verzija Linux kernela namenjena isključivo PC arhitekturi.

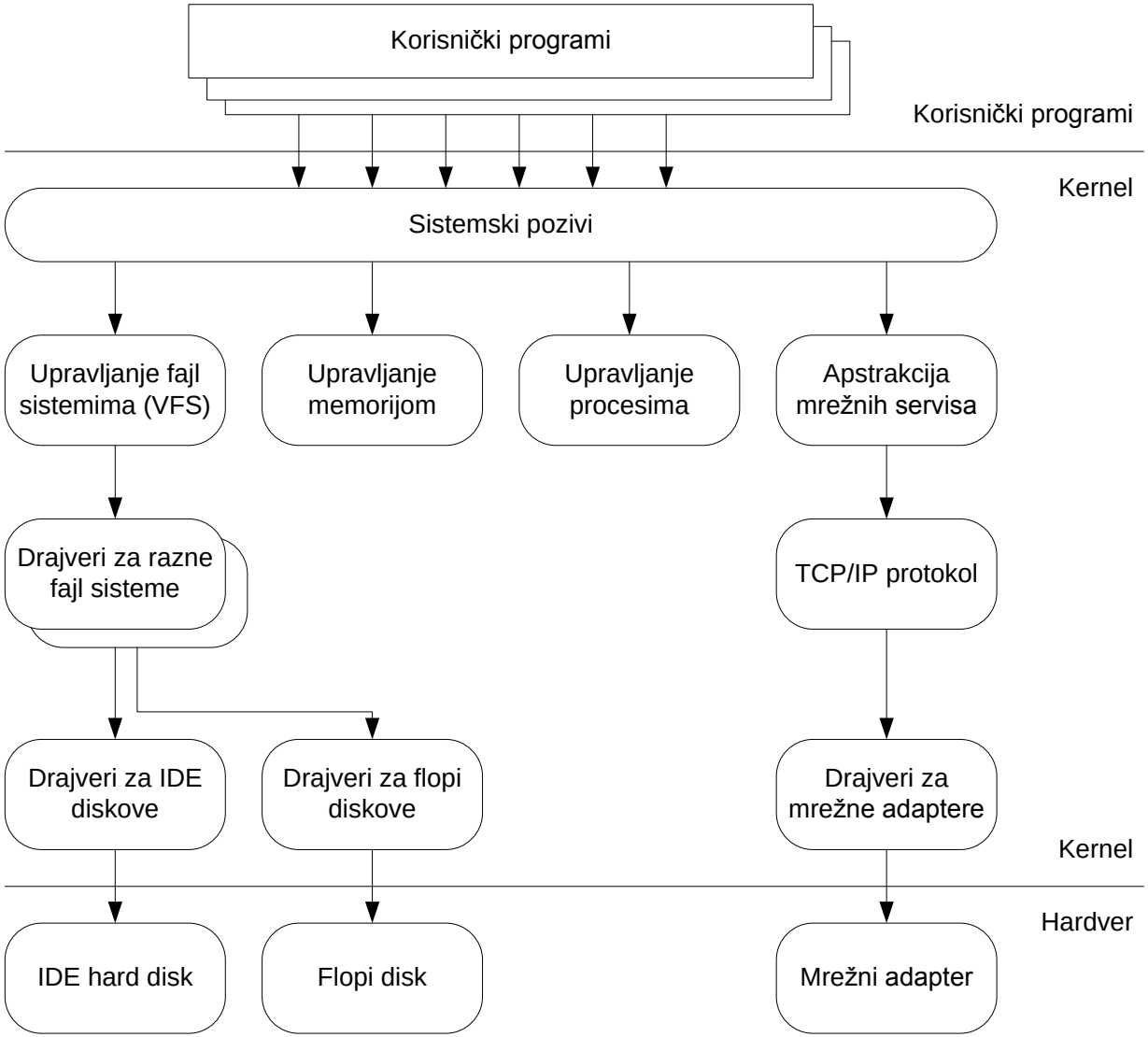
- **U verziji 2.0** (jun 1996) uvedena je podrška za više arhitektura (Motorola i Intel procesori, Sun Sparc i PowerMac sistemi), kao i podrška za višeprocesorsku arhitekturu (SMP). Dodatno, poboljšano je upravljanje memorijom i uvećane se performanse TCP/IP protokol steka, a ugrađena je i podrška za unutrašnje kernelske niti (internal kernel threads). Kernel je modularizovan, odnosno uvedena je mikro-kernel struktura sa izmenljivim drajverskim modulima (loadable kernel modules), a standardizovan je i konfiguracioni interfejs.

Struktura kernela Linux sistema

- Osnovu Linux sistema čine:
  - kernel
  - sistemske biblioteke
  - sistemski programi
- Kernel je odgovoran za najznačajnije funkcije operativnog sistema.
- Dve osnovne karakteristike kernela su:
  - kernel kod se izvršava u kernelskom modu u kome je jedino moguće pristupati svim komponentama hardvera
  - kompletan kernel kod i sve kelnel strukture podataka se čuvaju u istom adresnom prostoru (monolithic)

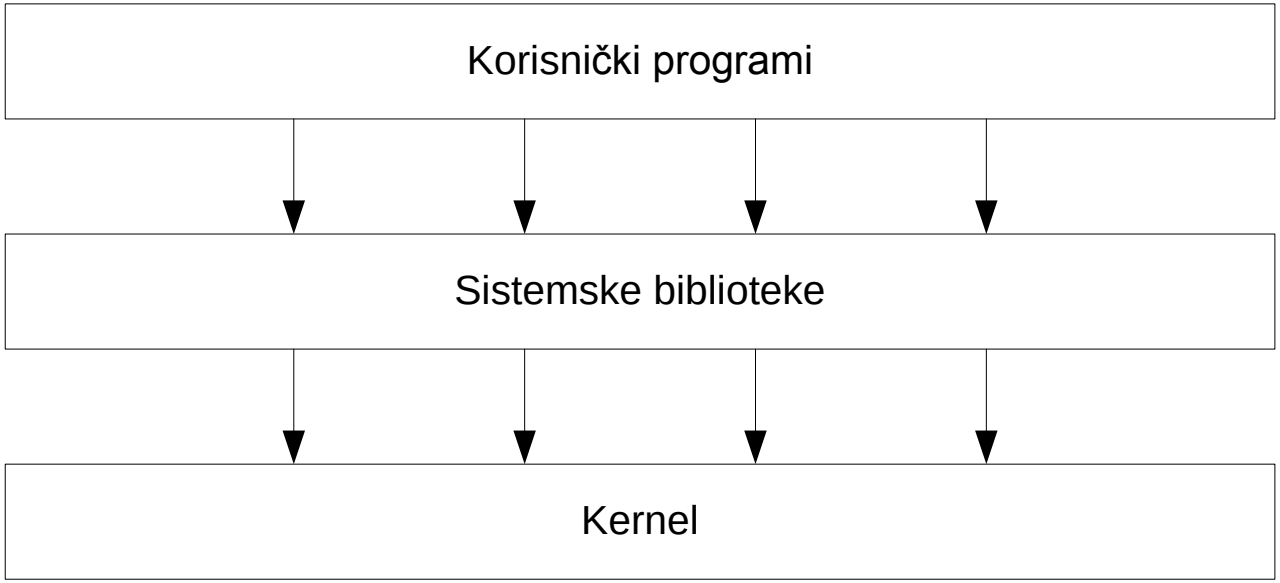
Opšti UNIX kernel - arhitektura

- Kod većine UNIX sistema, aplikacije se preko SC direktno obraćaju kernelu



Opšti Linux kernel - arhitektura

- Kod Linux sistema, sistemski pozivi se upućuju kernelu preko sistemskih biblioteka,
- koje definišu standardni set funkcija
- preko kojih aplikacije komuniciraju sa kernelom



Modularni kernel-uvod

- Moduli kernela su delovi kernelskog koda koji može da se prevede, napuni u memoriju ili izbaci iz memorije nezavisno od ostatatka kernela.

- Kernelski moduli implemetiraju:
  - drajvere za hardverske uređaje
  - novi sistem datoteka
  - mrežne protokole

- Moduli omogućavaju raznim programerima da napišu i distribuiraju drajvere koji ne moraju da prođu GPL licencu.

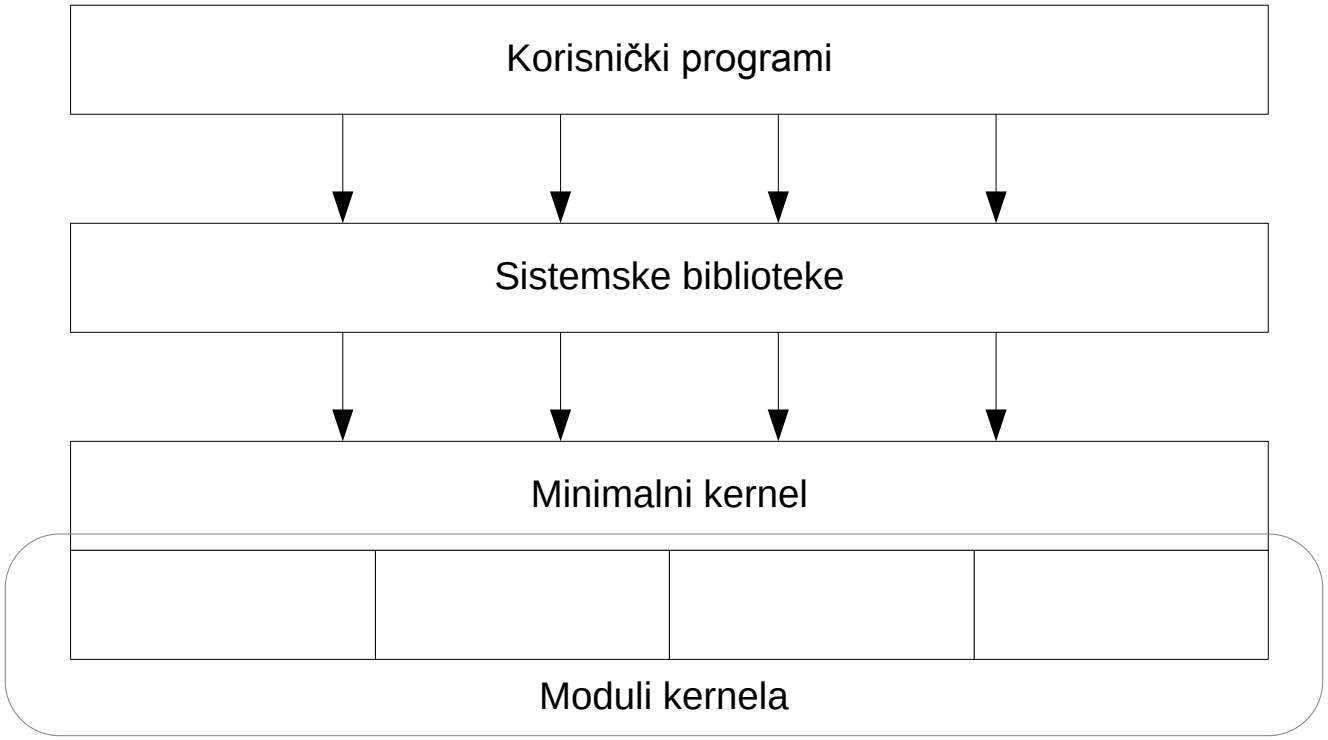
- Moduli kernela omogućavaju **micro-kernel arhitekturu**,
  - odnosno realizaciju minimalne stabilne konfiguracije kernela
  - bez dodatnih drajvera.

- Potrebni drajveri pune se u memoriju kao moduli kernela.

Komponente modula

- Module Linux kernela čine **3 komponente**:
  - **1. upravljanje modulom,** koja omogućava punjenje modula u kernelsku memoriju i komunikaciju modula sa ostatkom kernela, proveru da li je modul u memoriji i da li se koristi i izbacivanje modula iz memorije (pod uslovom da se modul ne koristi).
  - **2. registracija drajvera**, koja omogućava modulu da objavi ostatku kernela da je novi drajver u memoriji i da je raspoloživ za korišćenje. Kernel održava dinamičku tabelu drajvera, koji se pomoću posebnog seta programa mogu napuniti ili izbaci iz memorije u svakom trenutku,
  - **3. rezolucija konflikata**, odnosno mehanizam koji služi da spreči hardverske konflikte, tako što omogućava drajveru da rezerviše hardverske resurse (IRQ, DMA, ports) i time spreči druge drajvere ili autoprobe funkciju da ih koriste.
  - **#lsmod #modprobe #depmod #insmod #rmmod**

Struktura modularnog Linux kernela

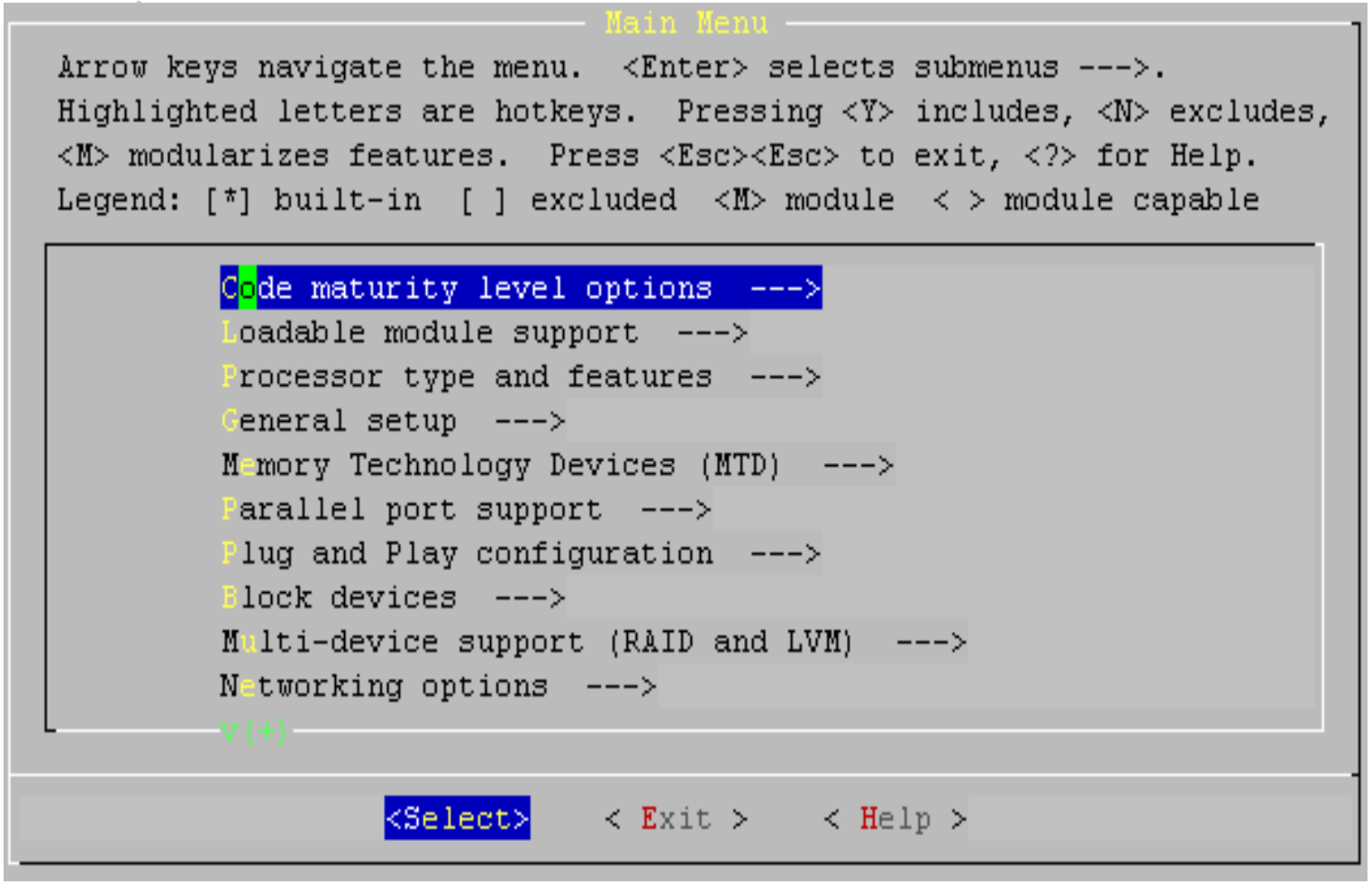


Konfiguracija, prevođenje i podešavanje Linux kernela verzija 2.4 na primeru RedHat Linux

- **Podešavanje putem menija** je **lakši i pregledniji način izrade konfiguracione datoteke (human-friendly interfejs za konfigurisanje).**
- **Okruženje sa menijima** se **pokreće** tako što se
  - na sistem prijavi root,
  - zatim pređe u **direktorijum** u kome se nalazi **izvorni kod kernela**
  - (na primer **/usr/src/kernel-source-2.4.19**) i
  - pokrene komandu **make menuconfig**.
  - **Alat za podešavanje** putem menija **zahteva** da je na sistem **instalirana biblioteka ncurses**.
- **# cd /usr/src/kernel-source-2.4.19**
- **# make menuconfig**
- Podešavanje putem menija **počinje spiskom kategorlja** u **kojima se biraju opcije kernela.:**
  - Za **navigaciju po menijima** koriste se **kursorski tasteri** (**strelice**),
  - za **ulazak u podmeni** taster **Enter**.
  - **Taster Tab** služi za prelazak u **deo ekrana** sa **dugmadima**
  - **Select** (**prikazuje podmeni opcija** pridruženih istaknutoj stavci),
  - **Exit** (vraća se na **prethodni meni**, završavajući podešavanje ukoliko je izabrano sa najvišeg nivoa) i
  - **Help** (prikazuje pomoćni ekran pun informacija o istaknutoj stavci menija).

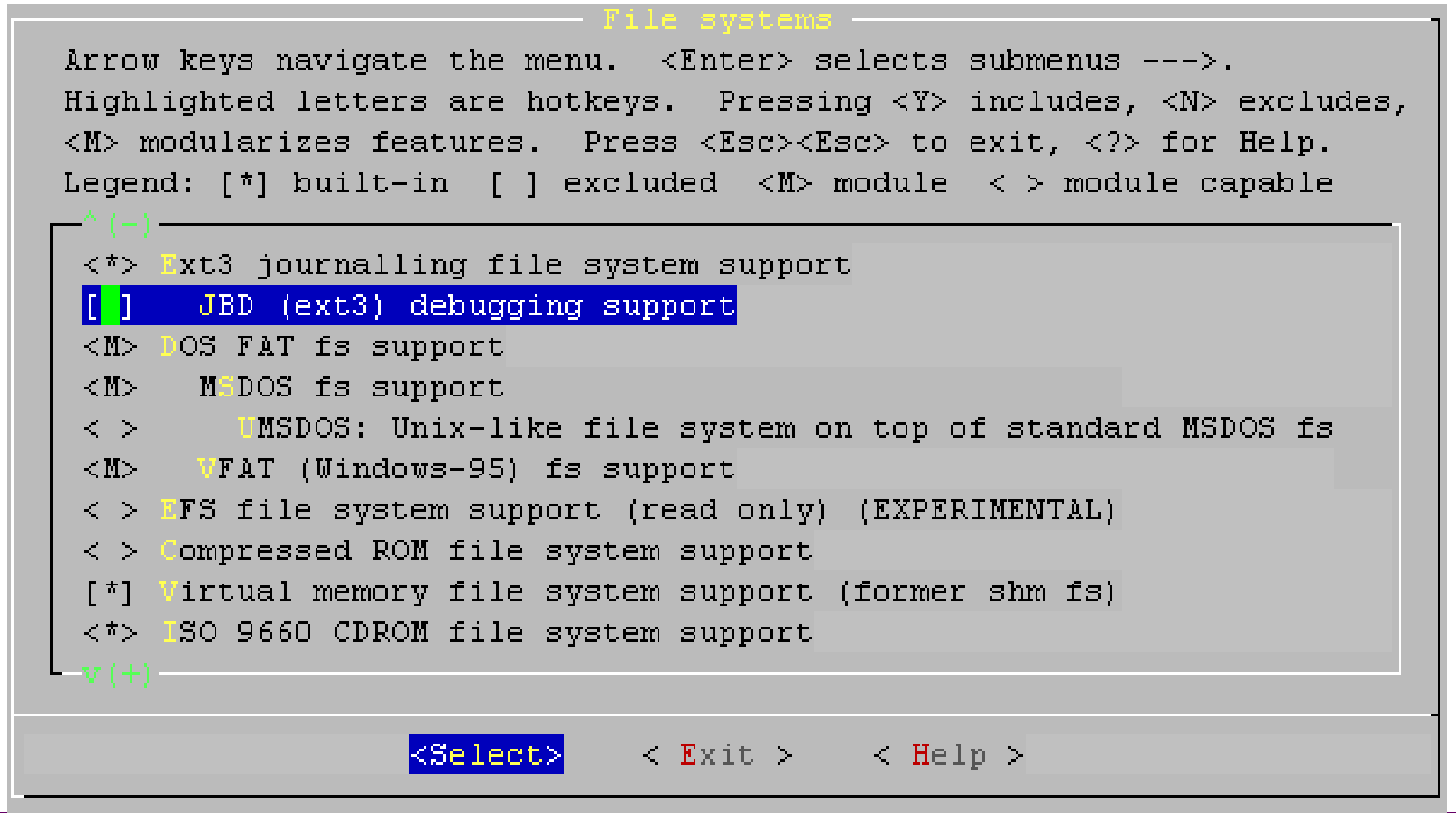
Konfiguracija, prevođenje i podešavanje Linux kernela verzija 2.4 na primeru RedHat Linux

- **make menuconfig**



Konfiguracija, prevođenje i podešavanje Linux kernela verzija 2.4 na primeru RedHat Linux

- **Stavke sa podmenijem** prikazane **imaju strelicu sa svoje desne strane**. **File Systems** iz glavnog menija otvara se **podmeni** prikazan na slici.

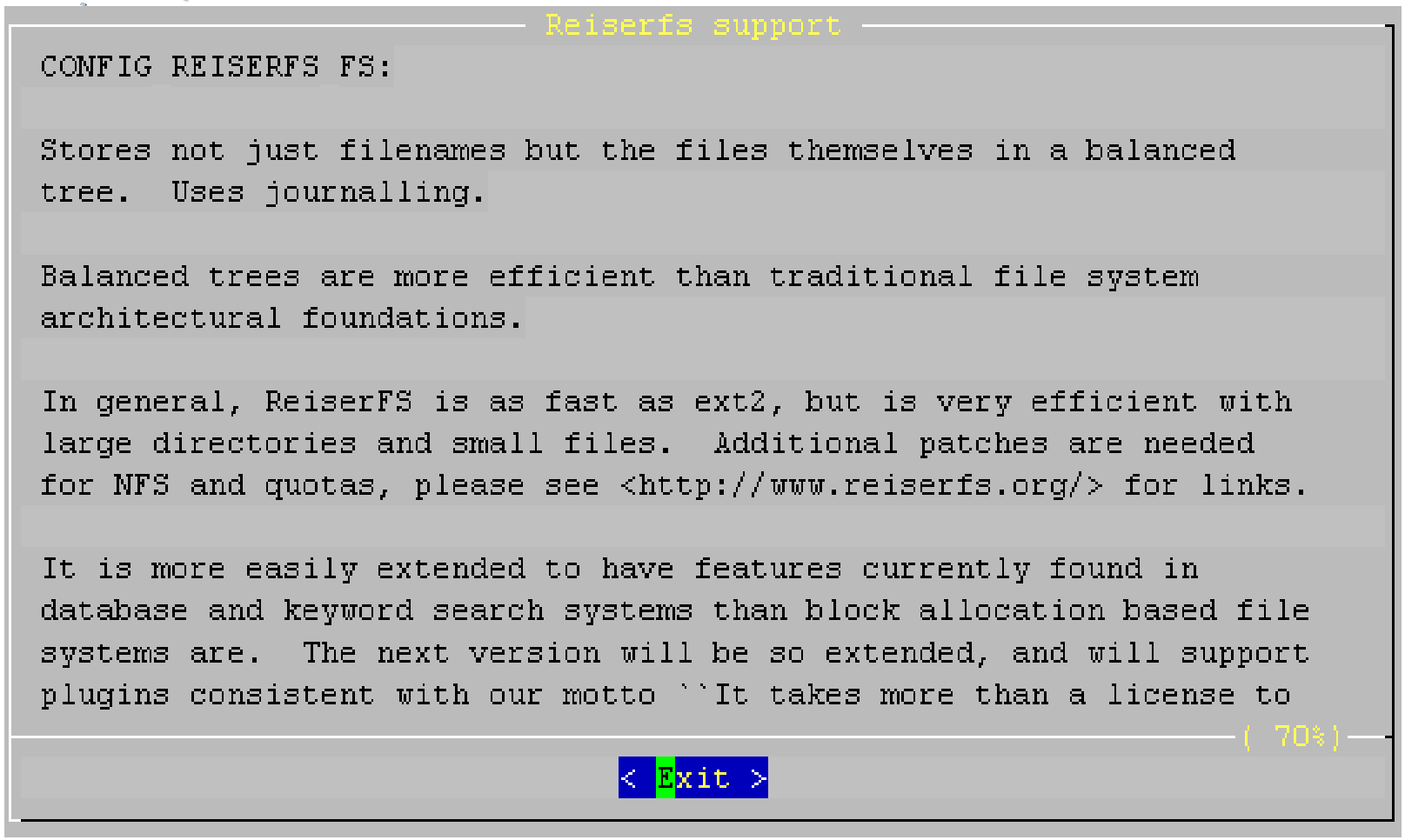


Konfiguracija, prevođenje i podešavanje Linux kernela verzija 2.4 na primeru RedHat Linux

- **Stavke menija** koje odreduju neku opciju jezgra, a **ne mogu biti modularne**, **sa leve strane imaju jednu** od **sledećih oznaka**:
  - [*] opcija je deo jezgra
  - [ ] opcija nije deo jezgra
- **Stavke menija koje odreduju neku opciju jezgra**, a **mogu biti deo jezgra** ili **realizovane kao modul**, **sa leve strane imaju jednu** od **sledećih oznaka:**
  - **< >** **opcija nije uključena kao deo jezgra**, niti napravijena kao modul koji može kasnije da se učita
  - **<*>** **opcija je deo jezgra** i samim tim je uvek deo sistema.
  - **<M>** **opcija je uključena kao modul**, ali nije deo samog jezgra. Modul je kasnije po potrebi može učitati ili ukloniti iz jezgra koje se izvršava.
  - **Oznake sa leve strane ukazuju na trenutno stanje opcije**.
- **Stanje se može izmeniti jednim od sledećih tastera**:
  - **Y (uključi** opciju u jezgro)
  - **M (uključi opciju kao modul)**
  - **N (**opcija se ne uključuje, niti realizuje kao modul).
- Dodatno, **pomoću tastera** ? se za tekuću opciju na ekranu prikazuje detaljan opis.

Primer za realizaciju
Konfiguracija, prevođenje i podešavanje Linux kernela verzija 2.4 na primeru RedHat Linux

- **Podešavamo Reiser FS**



Konfiguracija, prevođenje i podešavanje Linux kernela verzija 2.4 na primeru RedHat Linux

- **Nakon postavljanja svih opcija na željenu vrednost**, potrebno je iz glavnog menija odabrati dugme **Exit,** nakon čega će se na ekranu pojaviti pitanje da li **želite da snimite izmenjenu konfiguracionu datoteku**.

Konfiguracija, prevođenje i podešavanje Linux kernela verzija 2.4 na primeru RedHat Linux

- Prevođenje i testiranje novog jezgra
- **Nakon kreiranja nove konfiguracione** datoteke **potrebno je prevesti izvorni kod kernela**.
- **Zavisno od konfiguracije računara** (brzina procesora, količina memorije, tip i brzina diskova) i verzije kernela, **prevođenje i sklapanje jezgra** traje od **nekoliko minuta do nekoliko sati.**
- Kernel može biti
  - **nekomrimovan** (image) i
  - **komprimovan progrogramom gzip ili bzip2** (**zlmage, bzimage**), koji je manji, ali se sporije učitava.

- **Komande za prevođenje variraju** sa konkretnom Linux distribucijom, ali su u **svakom slučaju jednostavne i mogu izvršavati jedna za drugom zadate iz jedne komandne linije.** Komande se mogu **zadavati i odvojeno**, jedna za drugom.

Konfiguracija, prevođenje i podešavanje Linux kernela verzija 2.4 na primeru RedHat Linux

- Prevođenje i testiranje novog jezgra

- **Sledeće komande**
  - prevode izvorni kod,
  - prave novo jezgro i sklapaju sve module jezgra,
  - smeštajući ih u odgovarajuće sistemske direktorijume,
  - čime se obezbeđuje njihova dostupnost pomoću standardnih komandi za rad sa modulima:

- **# make dep; make clean; make zImage; make modules; make modules_install**

Delovi Linux kernela

- Linux kernel čini nekoliko značajnih komponenti:
  - upravljanje procesima
  - upravljanje memorijom
  - upravljanje sistemima datoteka (VFS)
  - apstrakcija mrežnih servisa
  - podrška za hardverske uređaje
  - podrška za različite sisteme datoteka
  - podrška za TCP/IP

Upravljanje procesima

- Linux koristi standardni UNIX proces mehanizam (fork)
- koji razdvaja kreiranje procesa i
- njegovo izvršenje u dve različite operacije:
  - 1. sistemski poziv fork, koji kreira novi proces
  - 2. sistemski poziv exec, koji izvršava program u resursima novostvorenog procesa
- Pod UNIX sistemom sve informacije koje operativni sistem mora
- čuvati da bi kontrolisao jedan proces
- predstavljaju **kontekst tog procesa.**
- Pod Linux operativnim sistemom, svaki proces je u potpunosti opisan sa **3 informacije:**
  - **identitet**
  - **okolina**
  - **kontekst**

Identitet i okolina procesa

- **Identitet procesa** obuhvata sledeće informacije:
  - **Identifikator procesa (Process ID - PID),** pomoću kog Linix kontroliše proces.
  - **Akreditivi (Credentials).** Svaki proces pripada jednom korisniku koji ima svoj user ID i jedan ili više grupnih IDs, koji određuju prava pristupa procesu u radu sa datotekama.
  - **Ličnost (Personality).** Ova informacija se ne koristi kod drugih UNIX sistema, a Linux svakom procesu dodeljuje lični identifikator koji može imati uticaja za neke sistemske pozive.

- **Okolina procesa** se nasleđuje od procesa roditelja. U okolinu procesa spadaju vektor argumenata koje proces roditelj prosleđuje programu i vektor okoline, odnosno lista promenljivih koje definišu okolinu procesa (environment).

Kontekst procesa

- Kontekst procesa je stanje procesa u datom trenutku vremena.
- Kontekst procesa čine sledeće komponente:
  - **kontekts za raspoređivanje** (scheduling context), koji služi za
  - efikasnu suspenziju ili ponovni start procesa. Obuhvata sve CPU registre, prioritet procesa i kernelski stek procesa.
  - **statistički kontekst**, koji sadrži informacije o resursima koje proses koriste u jednom trenutku, kao i kompletnu upotrebu resursa za vreme trajanja jednog procesa (accounting information)
  - **tabela datoteka** (file table), tj. polje ukazivača na kernelske strukture datoteka
  - **kontekst sistema datoteka** (file-system context)
  - **tabela za upravljanje signalima** (signal-handler table), koja definiše ukazivače na programe koji se pozivaju nakon određenog signala
  - **kontekst virtulene memorije** (virtual-memory context), koji potpuno opisuje korišćenje memorije od strane procesa

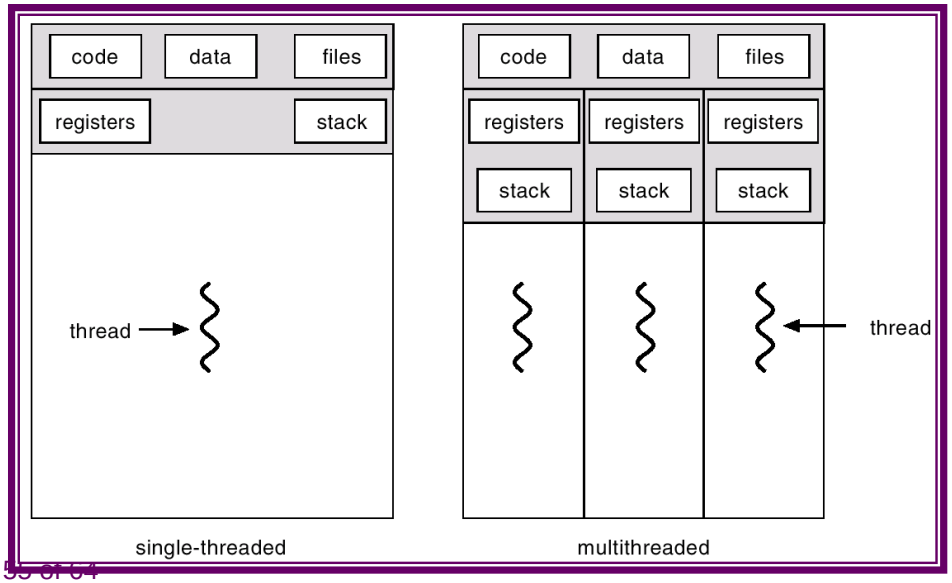
Procesi i niti

- Linux koristi istu internu reprezentaciju za procese i niti –
  - nit (thread) je jednostavno novi proces
  - koji deli adresni prostor roditelja.

- Za razliku od novog procesa koji pomoću sistemskog poziva fork formira novi kontekst sa unikatnim adresnim prostorom,

- nit nastaje pomoću sistemskog poziva **clone**,

- koji kreira novi kontekst, ali dozvoljava novom procesu da deli adresni prostor roditelja.



Dodeljivanje procesora procesima

- Linux koristi 2 algoritma za dodelu procesora procesima (process-scheduling algorithms):
- **1. time-sharing algoritam** za korektno raspoređivanje između procesa (fair preemptive scheduling).
- Dodela se vrši na osnovu prioriteta procesa koji definiše korisnik i kredita (efektivni prioritet) koji raste s porastom vremena čekanja na procesor po sledećoj **rekurzivnoj formuli:**

$$kredit = \frac{kredit}{2} + prioritet$$

- **2. real-time algoritam** za procese gde su apsolutni prioriteti mnogo značajniji od ravnomerne raspodele. Linux je ipak soft-real time operativni sistem.

Interprocesna komunikacija

- Interprocesna komunikacija obuhvata
  - obaveštavanje procesa o događaju i
  - prenos podataka s jednog procesa na drugi.
- Kao i UNIX sistem, Linux informiše procese u korisničkom režimu o događaju putem **signala.**
- Procesi u kernel modu umesto signala,
  - koriste specijalnu vrstu deljive memorije
  - (wait.queue struktura)
  - za interprocesnu komunikaciju.
- Za prosleđivanje podataka između procesa koristi se
- **1. pipe mehanizam**, koji omogućava jednosmernu razmenu podataka putem komunikacionog kanala koji proces nasleđuje od roditelja, i
- **2. deljiva memorija**, koja je brza i fleksibilna, ali zahteva sinhronizaciju

Upravljanje memorijom- fizička memorija

- Upravljanje memorijom obuhvata
  - upravljanje operativnom (RAM) memorijom i
  - upravljanje virtuelnom memorijom.

- Upravljanje fizičkom memorijom se bavi alokacijom i oslobađanjem stranice (pages, normal extent), grupe stranica (large extent) i malih memorijskih blokova (small extent).

- Upravljanje fizičkom memorijom se obavlja po sistemu drugova (Buddy heap).
  - Cela fizička memorija se deli na udružene blokove čije su veličine stepeni broja 2.
  - Blokovi se prema potrebi alokacije dalje razbijaju na manje blokove ili se parovi udružuju u veće celine.

Upravljanje memorijom – virtuelna memorija

- Sistem virtuelne memorije povećava ukupan adresni prostor koji je dostupan procesima –
  - sistem kreira stranicu virtuelne memorije na zahtev,
  - upravlja punjenjem te stanice u fizičku memoriju sa diska
  - i povratkom stranice na disk u swap prostor.

- Kada stranica mora da napusti memoriju i ode na disk, izvršava se takozvani page-out alogoritam, koji je na **Linux sistemu realizovan LFU konceptom (Least Frequently Used).**

- Novi virtuleni adresni prostor se formira nakon kreiranja novog procesa sistemskim pozivom fork i nakon izvršavanja novog programa sistemskim pozivom exec.

- Regioni virtulene memorije obuhvataju: fizičke stanice (frame) i swap prostor na disku.

Izvršavanje korisničkih programa

- Linux podržava brojne formate za punjenje i izvršavanje programa. Među njima svakako treba istaći stari UNIX format a.out i novi elf format, koji je maksimalno prilagođen konceptu virtulene memorije.

- ELF format se sastoji od zaglavlja, koje opisuje sekcije programa.
- Sekcije programa su po veličini prilagođenje veličini stanice virtulene memorije.

- Program kod koga su funkcije iz sistemske biblioteke direktno ugrađene u kod programa je program sa statičkim linkovanjem. Glavni nedostatak ovakvog linkovanja je povećanje veličine koda, jer svaki poziv funkcije iz biblioteke kopira celu funkciju u kod. Takođe čim je kod veći, veća je količina memorije potrebna za njegovo izvršavanje.

- Na drugoj strani, dinamičko linkovanje je efikasnije i modernije, sama funkcija se ne kopira u kod, a i manja količina memorije je potrebna za izvršenje.

Ulazno - izlazni sistem

- Linux deli uređaje u tri klase:
  - **blok uređaje (poput diskova i CD-ROM uređaja)**
  - **karakter uređaje (poput štampača)**
  - **mrežne uređaje**
- Svaki uređaj je predstavljen specijalnom datotekom (device node, device file) koja se nalazi u direktorijumu /dev root sistema datoteka.
- Kada korisnik upisuje podatke u datoteku koja predstavlja neki uređaj ili čita iz te datoteke, vrši se neka ulazno-izlazna operacija, odnosno sistem šalje ili prima podatke sa uređaja koji je predstavljen tom datotekom.
- Time se ukida potreba za postojanjem posebnih programa (a samim tim i posebnom metodologijom programiranja ulazno-izlaznih operacija) neophodnih za rad sa uređajima.
- Na primer, korisnik može da odštampa tekst na štampaču jednostavnom redirekcijom standardnog izlaza na datoteku /dev/lp1 koji predstavlja štampač:
- # **cat izvestaj.txt > /dev/lp1**
- Ova komanda će korektno odštampati datoteku izvestaj.txt ukoliko je ona u obliku koji štampač razume (npr. tekstualna datoteka).

Ulazno - izlazni sistem (dev)

- **Direktorijum /dev** nastaje prilikom instalacije Linux sistema i

- u njemu se nalaze sve specijalne datoteke,
- bez obzira na to da li je uređaj instaliran na sistem ili ne –
- postojanje datoteke /dev/sda ne znači da je na sistem instaliran SCSI disk.

- Postojanje svih datoteka olakšava proces instalacije novog hardvera, tj. oslobađa administratora sistema potrebe za kreiranjem specijalnih datoteka sa korektnim parametrima.

Sistemi datoteka i aktivno UNIX stablo

- Linux sistemi datoteka koriste hijerarhijsku strukturu stabla i semantku UNIX sistema datoteka.
- Interno, kernel sakriva detalje i upravlja različitim sistemima datoteka preko jednog nivoa apstrakcije, koji se naziva virtuleni sistem datoteka VFS.
- Aktivno Linux stablo datoteka čini jedan ili više sistema datoteka koji su montirani na odgovarajuće direktorijume preko kojih im se pristupa.
- Osnovu aktivnog stabla datoteka čini **korenski sistem datoteka** (root filesystem), čiji root direktorijum ujedno predstavlja i root direktorijum aktivnog stabla datoteka.
- Zavisno od hardverske konfiguracije i odluke administratora sistema, struktura aktivnog Linux stabla može biti jednostavna (aktivno stablo realizovano jednim sistemom datoteka), ili složena (aktivno stablo realizovano većim brojem sistema datoteka - root, /boot, /var, /usr, /home ...)

Mrežne strukture

- Umrežavanje je ključno područje funkcionalnosti Linux sistema.
- Linux koristi standardni TCP/IP protokol stek
  - kao osnovni komunikacioni protokol,
  - a dodatno podržava i brojne druge protokole
  - koji nisu uobičajeni za komunikaciju dva UNIX sistema (AppleTalk, IPX, Samba).
- Interno, umrežavanje pod Linux sistemom obuhvata tri softverska nivoa:
  - **socket interfejs**
  - **protokol drajvere**
  - **drajvere za mrežne kartice**