1. Sekvenca pokretanja operativnog sistema

Booting je proces pokretanja računara, koji se može inicirati hardverski (pritiskom na dugme) ili softverski (komandom).

Pri gašenju računara, potrebni podaci za ponovo pokretanje čuvaju se u trajnoj memoriji, a pri ponovnom uključivanju, operativni sistem nije odmah u RAM-u.

Prvo se izvršava mali program iz ROM memorije, zvan **bootstrap loader**, koji pokreće učitavanje drugih programa iz RAM-a.

Postoje **višefazni boot loaderi**.

Prva faza, **first-stage boot loader (FSBL)**, radi sa ograničenjem memorije (npr. 32KB kod IBM-kompatibilnih računara) i učitava neophodne podatke iz boot sektora diska (MBR(MasterBootRecord)).

Primjeri FSBL-a su coreboot i Das U-Boot.

Druga faza, **second-stage boot loader**, kao što su GNU GRUB ili BOOTMGR, učitava operativni sistem i omogućava korisničke opcije (npr. izbor OS-a ili režima).

Proces se završava kada je OS spreman za rad.

Kod mrežnog pokretanja, operativni sistem se prenosi preko mreže pomoću PXE(Preboot Execution Environmentt), bez potrebe za drajverima u ranim fazama.

---

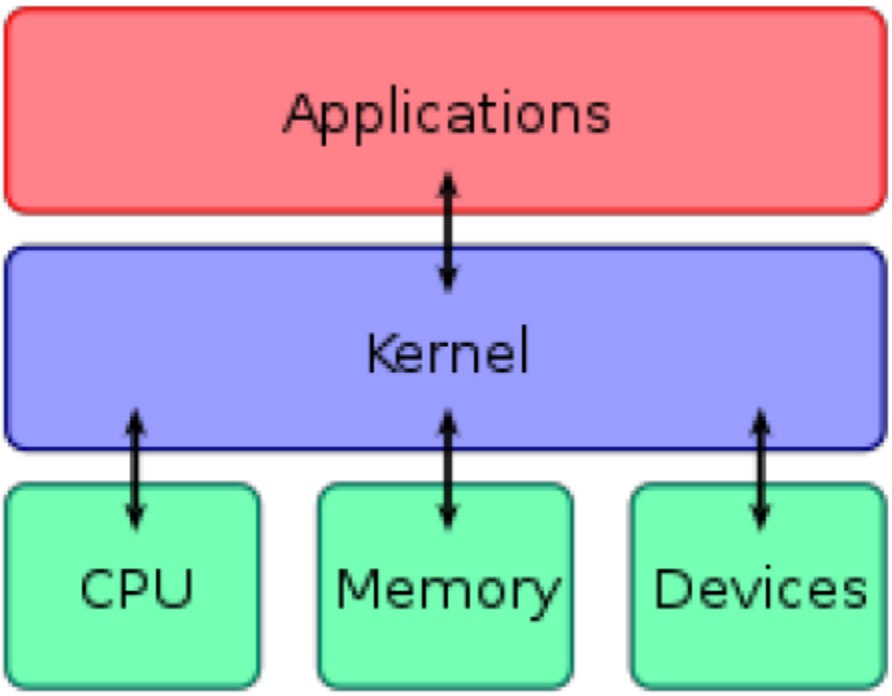
2. Kernel i tipovi kernela

Kernel je središnji dio operativnog sistema, sa potpunom kontrolom nad svim komponentama sistema.

On upravlja hardverom, memorijom i procesima te omogućava interakciju između softvera i hardvera.

Glavna uloga kernela je pretvaranje softverskih zahtjeva u uputstva za procesor.

Kernel se učitava prilikom pokretanja računara i ostaje aktivan sve vrijeme rada računara.



Znači:
1) Nacrtaš sliku!
2) Opišeš sve sa slike!

Tipovi kernela

1. **Monolitni kernel**

Monolitni kernel je tip arhitekture u kojem sve funkcije operativnog sistema rade u istom adresnom prostoru kao i kernel, što omogućava veliku brzinu.

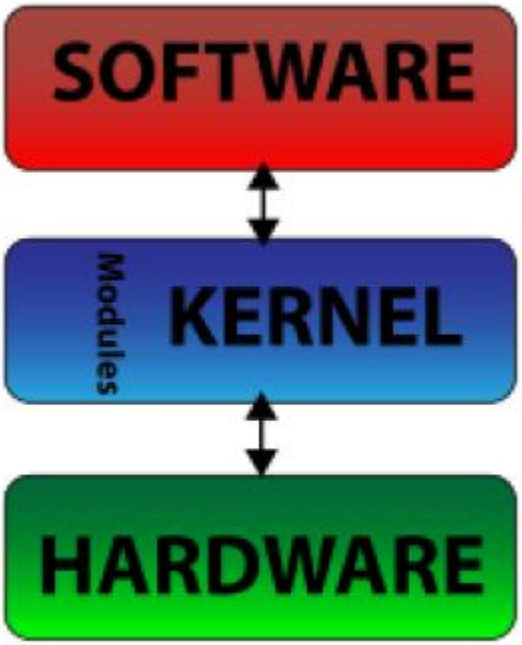
Međutim, glavni nedostatak je što greška u jednoj komponenti, npr. drajveru, može dovesti do rušenja cijelog sistema.

Zbog toga je održavanje velikih monolitnih kernela izazovno.

- **Prednosti**: Brzina, manji kod, manje bugova.

- **Nedostaci**: Teško debagovanje, greška može rušiti cijeli sistem, loša portabilnost.

- **Primjeri**: AIX, HP-UX, Solaris kernel, Linux, FreeBSD.



2. **Mikrokernel**

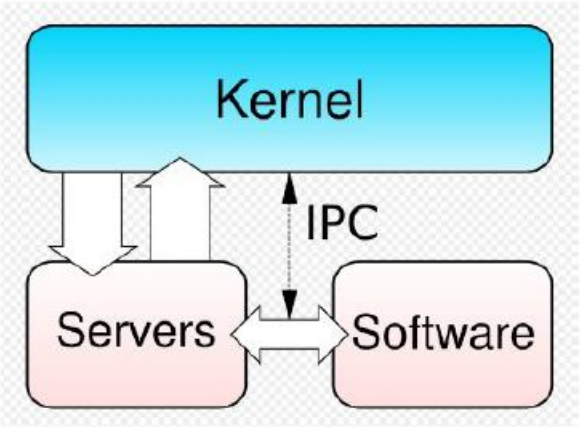
Mikrokernel premješta što više funkcionalnosti iz prostora kernela u korisnički prostor, ostavljajući samo osnovne funkcije u kernelu (upravljanje memorijom, multitasking, IPC).

Ovaj pristup omogućava lakše održavanje, ali može smanjiti performanse zbog velikog broja sistemskih poziva i promjena konteksta.

- **Prednosti**: Manja veličina, lakše održavanje, bolja modularnost.

- **Nedostaci**: Smanjena efikasnost, komplikovanije upravljanje procesima.

- **Primjeri**: MINIX, QNX, GNU Hurd.



3. **Hibridni kernel**

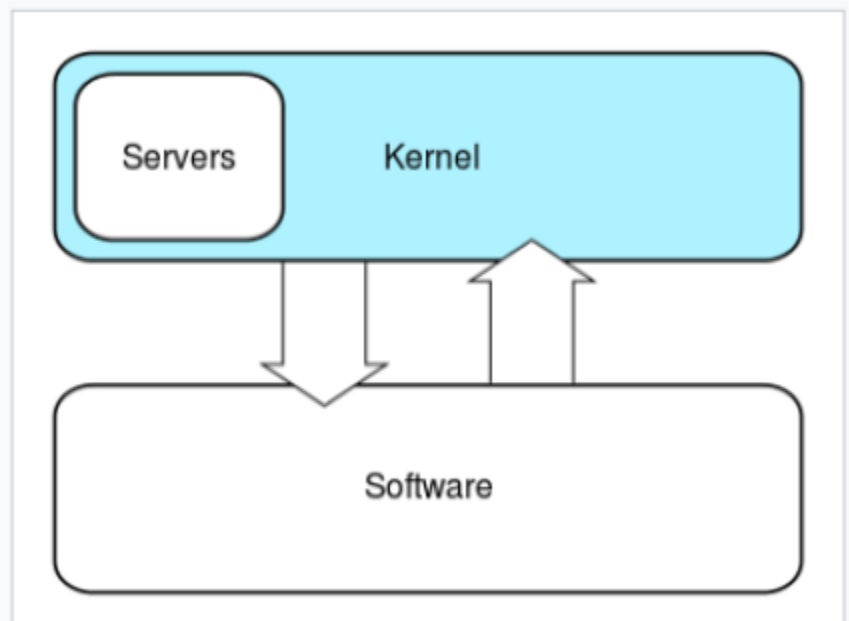
Hibridni kernel je kombinacija monolitnog i mikrokernela, sa ciljem da iskoristi prednosti oba.

Dodatni kod se smješta u prostor kernela kako bi se poboljšale performanse, ali se zadržava modularnost mikrokernela.

- **Prednosti**: Brže testiranje bez potrebe za restartom, lakša integracija novih funkcionalnosti.

- **Nedostaci**: Mogući sigurnosni problemi zbog više interfejsa, održavanje modula može biti složenije.

- **Primjeri**: Windows NT, macOS (XNU kernel).



4. **Nanokernel**

Nanokernel delegira gotovo sve funkcije drajverima uređaja, čime kernel postaje još manji od mikrokernela.

5. **Exokernel**

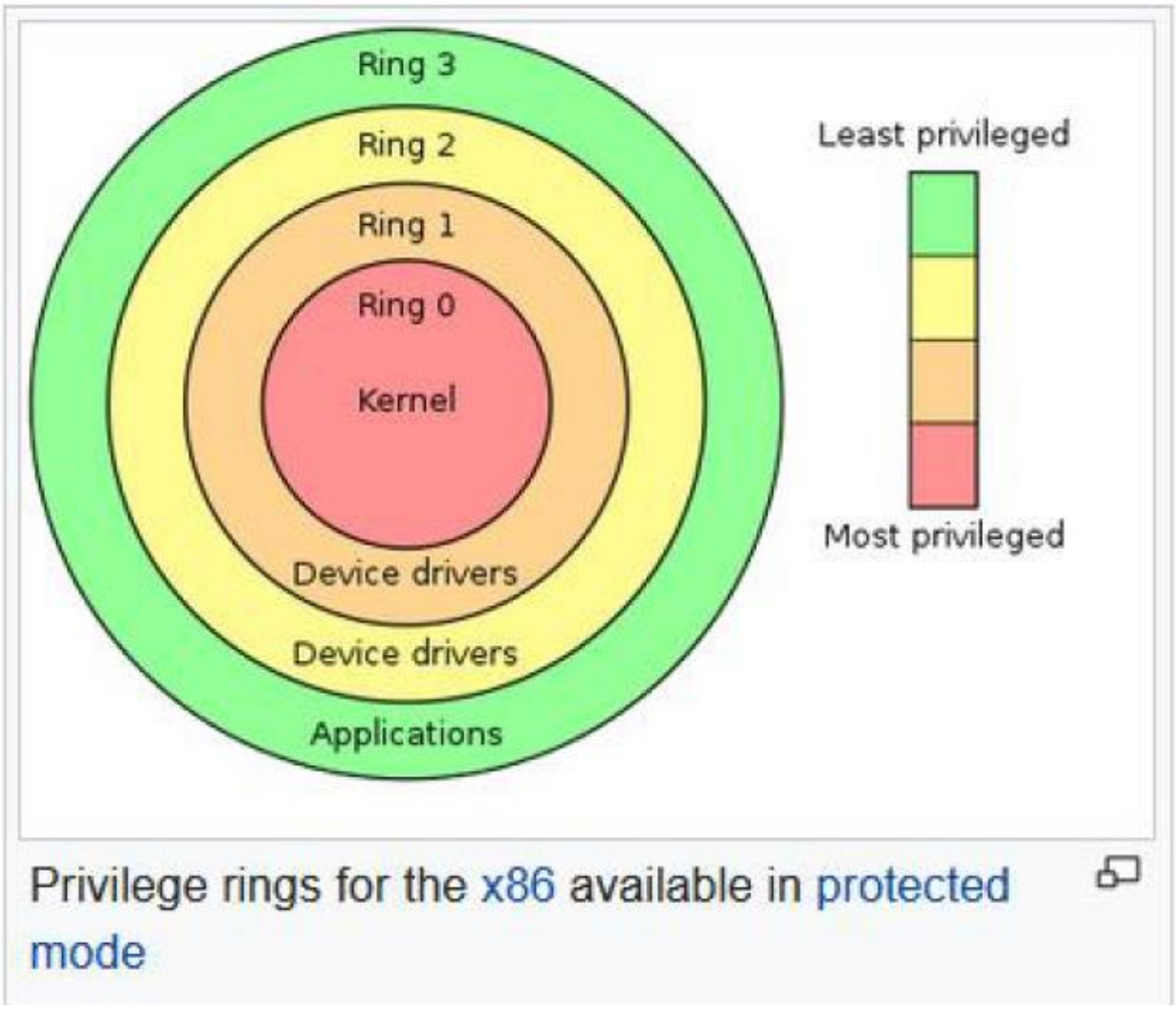
Exokernel je eksperimentalni tip kernela koji pruža minimalnu funkcionalnost za upravljanje hardverom, omogućavajući softveru direktnu kontrolu nad hardverom.

Njegova glavna prednost je fleksibilnost, jer omogućava različite biblioteke operativnih sistema za različite aplikacije.

Svaki od ovih tipova kernela pruža specifične prednosti i nedostatke, zavisno od potrebe za performansama, sigurnošću i modularnošću.

---

3. Prstenovi i Privilegije



Privilege rings for the x86 available in protected mode

Hijerarhijski zaštitni domeni ili **zaštitni prstenovi** su sigurnosni mehanizmi koji se koriste u računarstvu radi zaštite podataka i funkcionalnosti.

Oni pomažu u **poboljšanju otpornosti na greške** i štite od malicioznih radnji.

Prstenovi predstavljaju **nivo privilegija** unutar računalne arhitekture i postavljeni su hijerarhijski,

počevši od najprivilegovanijeg (prsten 0) do najmanje privilegovanog (prsten 3 ili veći).

- **Prsten 0**: Ima **najviše privilegije**, što znači da može direktno interagovati s fizičkim hardverom kao što su procesor i memorija.

Obično je u ovom prstenu pokrenut **operativni sistem** (OS) ili njegov kernel.

- **Prsten 3**: Ima **najmanje privilegije**, i obično je namenjen korisničkim aplikacijama.

Programi pokrenuti u ovom prstenu ne mogu direktno pristupiti hardveru bez specifičnih dozvola.

Princip rada:

Spoljašnji prstenovi (npr. prsten 3) mogu imati ograničen pristup resursima unutrašnjih prstenova (npr. prsten 0), ali samo na unapred definisan način.

Ovo pomaže u održavanju **sigurnosti sistema**,

jer npr. **spyware** pokrenut kao korisnička aplikacija ne može upaliti web kameru bez dozvole, jer kamera zahteva pristup drajverima u prstenu 1. Slično tome, web browser mora tražiti dozvolu za pristup mrežnim resursima.

Korišćenje prstenova u operativnim sistemima:

- Većina operativnih sistema koristi **dva prstena** – prsten 0 (za kernel) i prsten 3 (za korisničke aplikacije).

Ovaj model koriste Windows 7, Linux, macOS, i drugi.

- Neki sistemi, poput **OS/2** koriste 3 prstena, dok **OpenVMS** koristi 4, a **Multics** čak 8 prstenova.

- Sistemi kao što su DOS ili mnogi ugrađeni uređaji rade isključivo u **supervizorskom režimu** (najviši nivo privilegija).

Supervizorski režim:

Supervizorski režim omogućava izvršavanje svih instrukcija, uključujući i one koje su privilegovane. OS je obično pokrenut u ovom režimu dok su aplikacije u korisničkom režimu.

Kada korisnički program želi da izvrši privilegovanu funkciju, upućuje **sistemski poziv** prema OS-u, koji zatim obavlja zadatak i vraća kontrolu nazad programu.

x86 Arhitektura:

x86 procesori imaju **četiri nivoa privilegija** (prsten 0 do prsten 3):

- **Prsten 0**: Programi u ovom prstenu imaju potpuni pristup sistemu.

- **Prsten 3**: Programi u ovom prstenu nemaju pristup kritičnim resursima (poput memorije i I/O portova) bez dozvole.

Moderni OS-ovi (Windows, macOS, Linux, iOS, Android) koriste **paging mehanizam** sa jednim bitom (S/U bit), kojim se resursi kvalifikuju ili kao **supervizorski** ili kao **korisnički**.

Virtuelizacija:

Sa pojavom **virtuelizacije**, moderni procesori (Intel i AMD) nude podršku za **virtuelne mašine**.

Ovi procesori dodaju novi prsten, poznat kao **„Ring -1“**,

za hipervizore koji omogućavaju **kontrolu nad gostujućim operativnim sistemima** bez direktnog uticaja na host sistem.

Zaključak:

Upotreba zaštitnih prstenova je ključna za **sigurnost i stabilnost** računarskih sistema.

Iako mnogi sistemi koriste samo dva nivoa (kernel i korisnički),

prstenovi omogućavaju precizniju kontrolu nad time ko ima pristup resursima i kako se ti resursi koriste.

4.MBR i GPT

Master Boot Record (MBR) je stari tip boot sektora, predstavljen 1983. godine, koji se nalazi na početku uređaja za skladištenje podataka.

MBR sadrži podatke o organizaciji diska i izvršni kod (boot loader) koji pokreće operativni sistem.

Međutim, MBR ima ograničenje od 2TiB(2^{32}x500[B]) po particiji, što ga čini zastarelim za moderne potrebe.

GUID Partition Table (GPT) je moderniji standard za organizaciju particija, koji koristi univerzalne identifikatore (GUID).

GPT je deo UEFI standarda, ali se može koristiti i sa klasičnim BIOS-om. Za razliku od MBR-a, GPT nema ograničenje na veličinu particija,

što ga čini pogodnijim za moderne računare. Svi moderni OS-ovi podržavaju GPT, dok neki, kao macOS i Windows, zahtevaju EFI(Extensible Firmware Interface) firmware za boot sa GPT particija.

---

5. Procesi i model procesa

Proces je centralni koncept svakog operativnog sistema, predstavljajući instancu programa u izvršavanju.

Procesi se konkurentno izvršavaju na jednom procesoru, stvarajući iluziju paralelizma, iako je to zapravo **pseudoparalelizam, jer CPU prelazi sa jednog procesa na drugi.**

Ovo omogućava **multiprogramiranje**, gde CPU naizmenično koristi vreme za različite procese.

Proces je aktivnost koja uključuje program, ulaze, izlaze i stanje (kao što su registri i varijable). Razlika između procesa i programa je u tome što je proces aktivan entitet,

dok je program pasivan – nalazi se na disku. Više procesa može koristiti isti program, ali svaki proces ima svoje stanje i resurse.

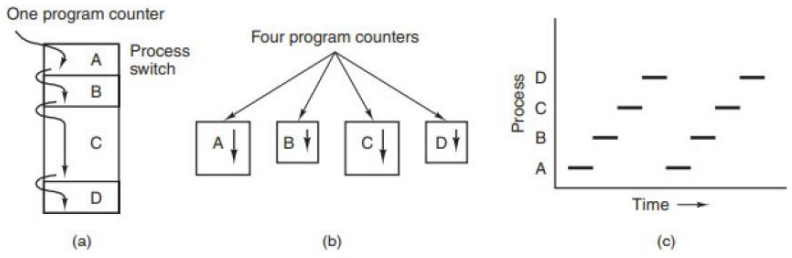


Figure 2-1. (a) Multiprogramming four programs. (b) Conceptual model of four independent, sequential processes. (c) Only one program is active at once.

Na slici (a) vidimo multiprogramiranje četiri procesa u memoriji, na slici (b) vidimo 4 procesa, svaki sa svojom kontrolom toka (vlastitim logičkim PC) i svaki se izvršava nezavisno od ostalih. Naravno, postoji samo jedan stvarni PC, pa svaki put kada se neki proces izvršava, njegov logički PC se upisuje u stvarni PC. Kada se trenutno izvršavanje završi, fizički PC se smješta u logički PC. Na slici (c) vidimo da su svi procesi napredovali u izvršavanju, ali se u svakom trenutku izvršava samo jedan proces.

---

6. Kreiranje i terminiranje procesa

U operativnim sistemima, kreiranje i terminiranje procesa su ključne funkcije.

U jednostavnim sistemima, svi procesi mogu biti kreirani pri pokretanju sistema.

U sistemima opšte namene, procesi se dinamički kreiraju tokom izvršavanja.

Procesi mogu biti kreirani iz različitih razloga, uključujući sistemsku inicijalizaciju, korisnički zahtev ili potrebu za obradom batch poslova.

Procesi se dele na **foreground** (interaktivne) i **background** (pozadinske), pri čemu su pozadinski često zaduženi za sistemske funkcije, poput web servera, printanja ili e-mail servisa. Ti pozadinski procesi se nazivaju **daemoni**.

Kreiranje novih procesa omogućava efikasniju organizaciju obrade zadataka, posebno kada procesi moraju međusobno komunicirati ili deliti resurse.

U UNIX-u, novi procesi se kreiraju sistemskim pozivom **fork**, koji klonira postojeći proces. Nakon toga, procesi roditelj i dete dele istu memorijsku sliku, ali svaki ima svoj adresni prostor.

Procesi često koriste sistemski poziv **execve** da učitaju novi program. U Windows-u, funkcija **CreateProcess** kreira novi proces i istovremeno učitava program u njega.

Procesi se obično terminiraju iz četiri razloga: normalan završetak, greška u izvršavanju, fatalna greška, ili kada ih terminira drugi proces.

U UNIX-u, poziv **kill** služi za terminiranje procesa, dok je u Windows-u to **TerminateProcess**.

Proces koji traži terminiranje drugog mora imati odgovarajuće ovlasti.

Po završetku procesa, svi resursi koje je koristio se oslobađaju.

---

7. Stanja procesa –prepričaj sliku!

Procesi u operativnim sistemima mogu se nalaziti u tri osnovna stanja: **running** (izvršava se), **ready** (spreman za izvršavanje) i **blocked** (blokiran, čeka na neki događaj).

Proces u stanju "running" koristi CPU, dok je proces u stanju "ready" spreman da se izvršava, ali čeka na slobodan CPU.

"Blocked" stanje se dešava kada proces ne može nastaviti jer čeka na neki spoljašnji događaj, poput unosa podataka.

Prelazi između ovih stanja su uslovljeni različitim situacijama. Proces može preći iz "running" u "blocked" kada čeka na ulaz.

Raspoređivač procesa može prekinuti izvršavanje procesa, što ga stavlja u stanje "ready" kako bi drugi proces dobio CPU.

Kada događaj na koji proces čeka bude završen, proces prelazi iz "blocked" u "ready".

Na taj način, procesi neprestano prelaze između ovih stanja u zavisnosti od resursa i događaja koji čekaju.

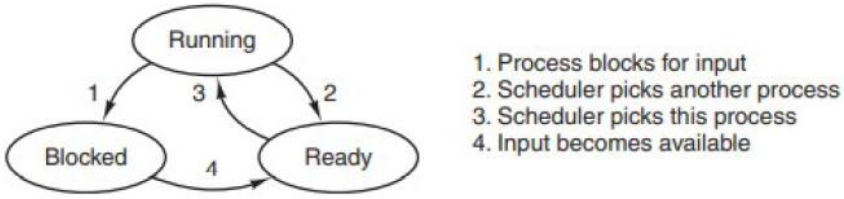


Figure 2-2. A process can be in running, blocked, or ready state. Transitions between these states are as shown.

---

8. Niti i upotreba niti

U operativnim sistemima, niti (threads) omogućavaju pokretanje više sekvencijalnih aktivnosti unutar jednog procesa, što ih čini efikasnijim u odnosu na tradicionalne procese.

Dok svaki proces ima svoju memoriju i jednu kontrolnu nit, korišćenje više niti unutar istog adresnog prostora omogućava da se više zadataka obavlja istovremeno bez potrebe za kreiranjem zasebnih procesa.

Niti su posebno korisne jer su "lakše" od procesa, što znači da ih je brže kreirati i uništiti, a takođe omogućavaju preklapanje ulazno-izlaznih (I/O) operacija i procesiranja podataka, čime se poboljšavaju performanse.

Niti se često koriste u aplikacijama koje moraju obavljati više zadataka paralelno.

Na primer, u tekstualnim procesorima, jedna nit može interagovati s korisnikom, dok druga vrši reformatiranje teksta u pozadini. Ovo omogućava brži odziv aplikacije bez čekanja da se cijeli dokument reformatira.

Također, treća nit može automatski čuvati dokument svakih nekoliko minuta, čime se obezbjeđuje sigurnost podataka bez prekidanja korisničkog rada.

Na sistemima sa više procesora, niti omogućavaju stvarni paralelizam, što znači da se zadaci mogu stvarno izvršavati istovremeno, a ne samo kvazi-paralelno.

Takođe su korisne za aplikacije koje rade sa velikim količinama podataka.

Na primjer, aplikacija može biti podijeljena u tri niti: jedna za unos podataka, druga za obradu i treća za izlaz podataka.

Svaka nit može raditi svoju funkciju istovremeno, čime se ubrzava obrada i smanjuje čekanje.

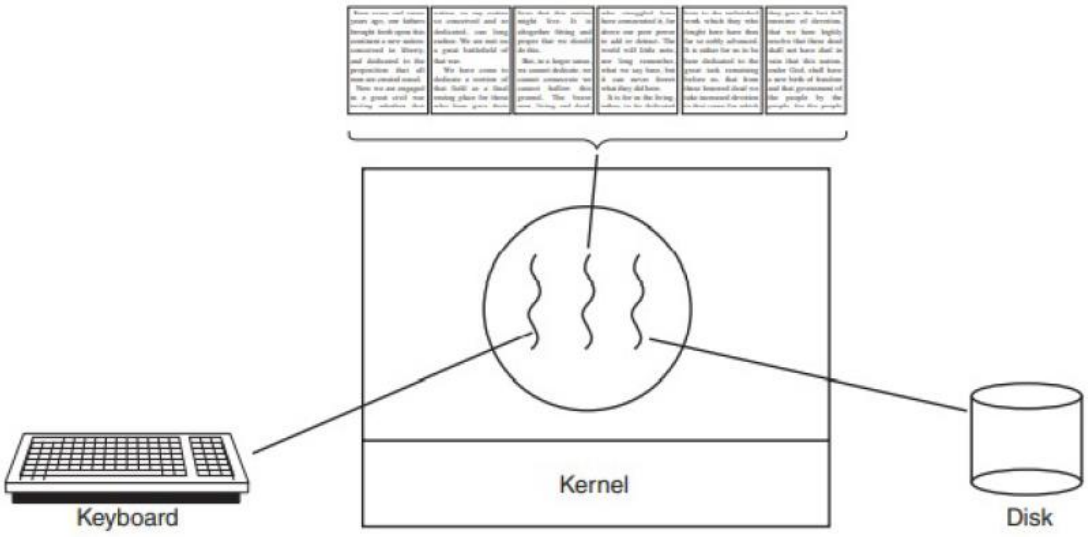


Figure 2-7. A word processor with three threads.

---

9. Niti- Klasični model

Klasični model niti razlikuje niti od procesa po tome što niti omogućavaju paralelno izvršavanje unutar jednog procesa, dijeleći resurse poput memorije, otvorenih fajlova, i signala.

Niti su "lakši" od procesa jer ne zahtijevaju potpuno odvajanje resursa, već djeluju unutar zajedničkog okruženja, što im omogućava bržu kreaciju i manju potrošnju resursa.

Multithreading omogućava izvršavanje više niti unutar jednog procesa, slično multiprogramiranju, gdje se CPU prebacuje između različitih niti.

Za razliku od procesa, niti nisu u potpunosti nezavisne – dijele isti adresni prostor i globalne varijable, što može dovesti do potencijalnih problema.

Na primjer, jedna nit može nehotice izmijeniti stek druge niti ili promijeniti zajedničke resurse poput fajlova.

Ipak, niti omogućavaju efikasnije upravljanje resursima, jer mogu dijeliti resurse i raditi zajedno na izvršavanju zadataka.

Niti se, kao i procesi, mogu nalaziti u stanjima kao što su "running" (trenutno se izvršava), "blocked" (čeka na događaj) ili "ready" (spremna za izvršavanje).

Kreiranje novih niti je moguće pozivom funkcija iz biblioteka, a završavanje niti se postiže pozivom funkcije "thread exit". Takođe, postoji mogućnost da jedna nit sačeka drugu da završi koristeći "thread join", što omogućava sinhronizaciju između niti.

Međutim, multithreading donosi izazove, poput onih u vezi sa sistemskim pozivima kao što je "fork".

Kada se proces forkuje, postavlja se pitanje da li dijete proces nasleđuje niti roditelja i kako se rešavaju situacije kada niti čekaju na različite događaje.

Slično, problemi se mogu pojaviti kada dvije niti istovremeno pokušaju pristupiti resursima poput fajlova ili memorije. Zbog toga je pažljivo planiranje neophodno kako bi multithreaded programi ispravno funkcionisali.

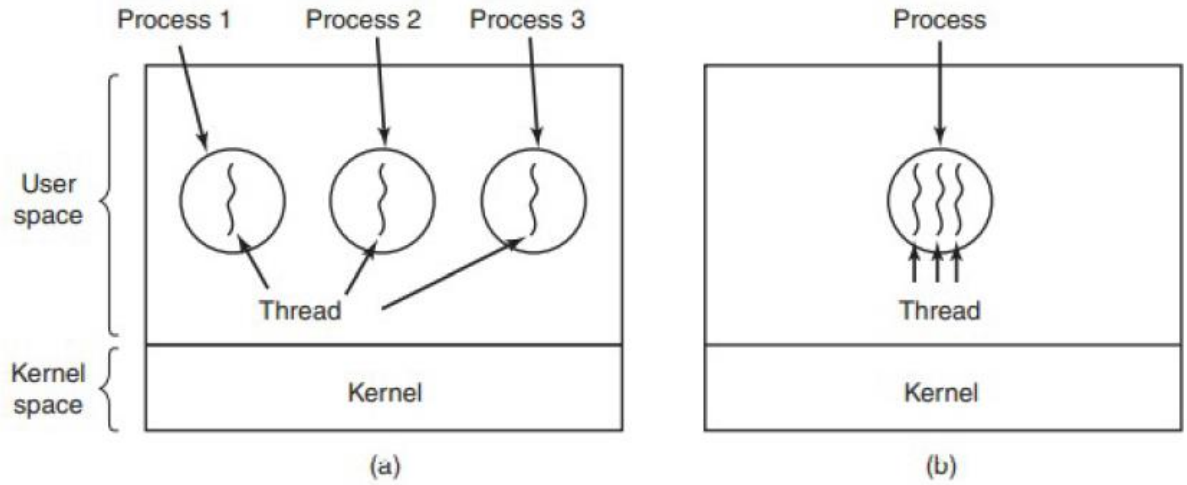


Figure 2-11. (a) Three processes each with one thread. (b) One process with three threads.

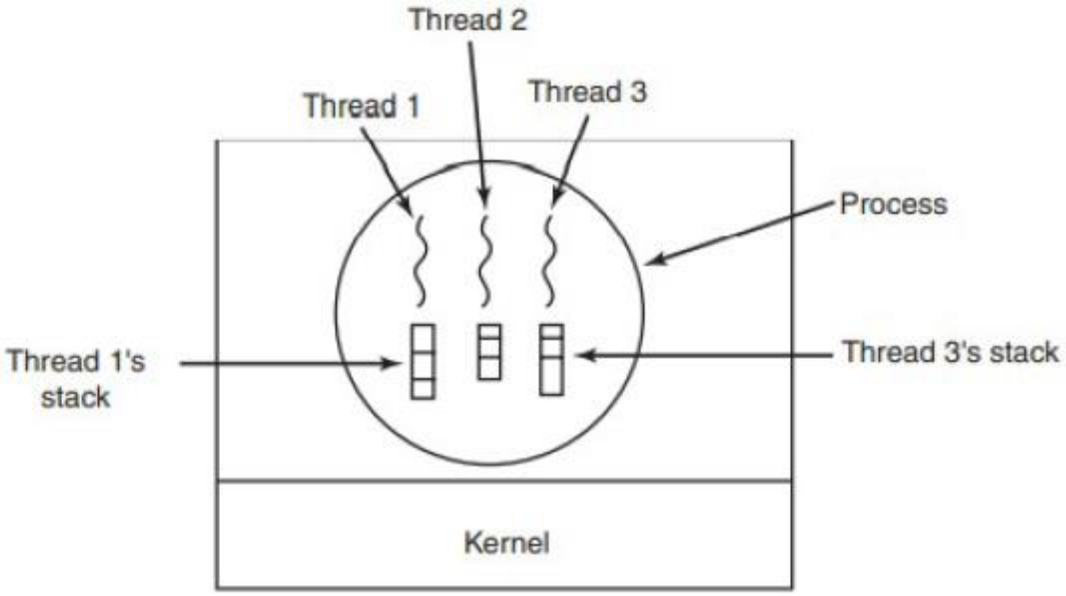


Figure 2-13. Each thread has its own stack.

---

10. User i kernel-space niti, hibridne i pop-up niti

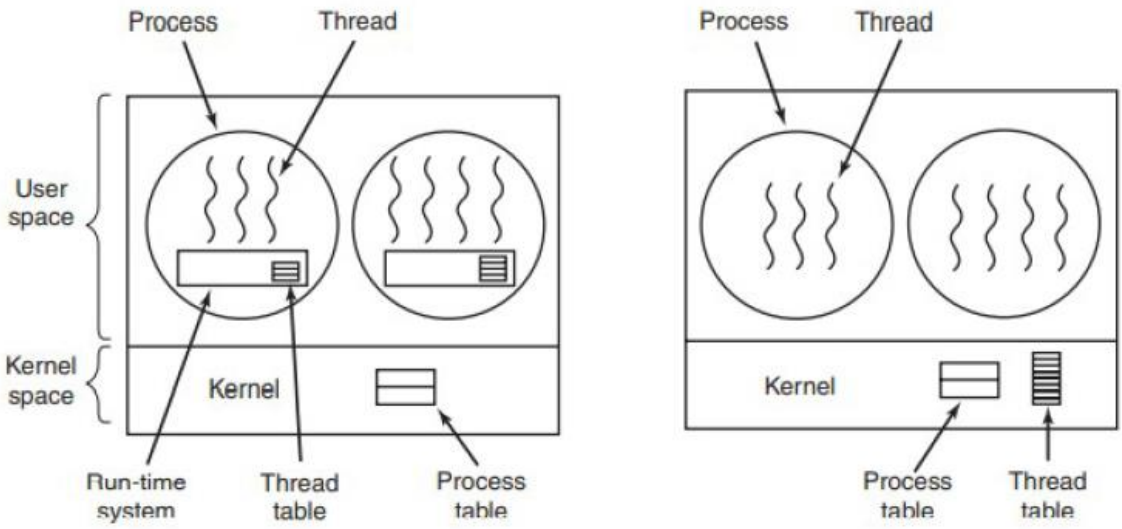


Figure 2-16. (a) A user-level threads package. (b) A threads package managed by the kernel.

U implementaciji niti postoje dva glavna prostora: korisnički prostor i prostor kernela, s mogućnošću hibridne implementacije.

Korisnički pristup podrazumeva da kernel ne prepoznaje niti, tretirajući ih kao procese sa jednim threadom.

Prednost ovog pristupa je brže raspoređivanje niti, jer svaki proces može koristiti vlastiti algoritam raspoređivanja.

Takođe, ne zahteva dodatni prostor u tabelama niti u kernelu.

Međutim, glavni nedostatak je to što blokirajući sistemski pozivi mogu zaustaviti sve niti, jer kernel nije svestan njihovog postojanja.

S druge strane, u pristupu sa kernel niti, kernel upravlja svim nitima, vodeći evidenciju u jednoj tabeli.

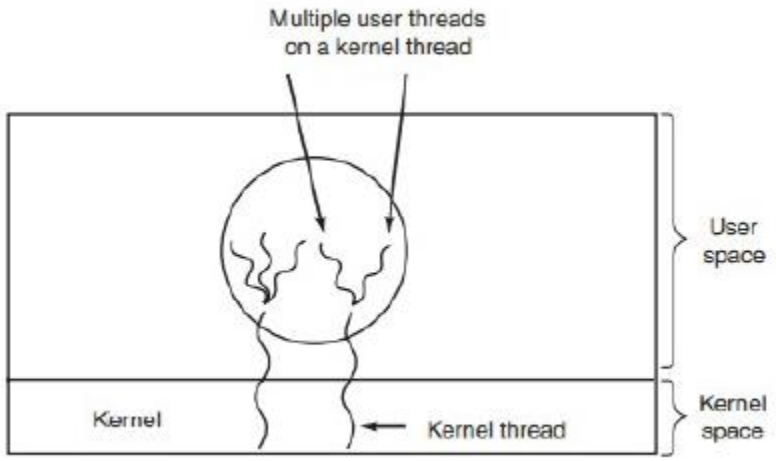


Figure 2-17. Multiplexing user-level threads onto kernel-level threads.

Ovaj pristup omogućava da kada jedna nit blokira, kernel može da izvršava druge niti, što poboljšava efikasnost.

Iako su kernel niti robusnije, troškovi sistemskih poziva mogu biti visoki.

Takođe, postavljaju se pitanja o signalima i grananju niti.

Hibridne implementacije kombinuju prednosti korisničkih i kernel niti, gde se kernel niti koriste kao osnova, a korisničke niti se raspoređuju na njih.

Ovaj pristup omogućava programerima da biraju broj kernel niti i korisničkih niti koje će se koristiti, čime se obezbeđuje veća fleksibilnost.

Postoji i koncept aktivacija raspoređivača, koji nastoji da zadrži karakteristike kernel niti uz bolje performanse.

Korisničke niti ne bi trebale da prave posebne neblokirajuće pozive, a kada dođe do blokiranja, kernel obaveštava runtime sistem koji može preraspodeliti niti.

Pop-up niti su posebna vrsta niti koja se koristi u distribuiranim sistemima, a kreiraju se kada poruka stigne.

Ove niti nemaju istoriju i brzo se kreiraju, čime se smanjuje kašnjenje obrade.

Međutim, potreban je pažljiv dizajn kako bi se odredilo u kojem prostoru (korisničkom ili kernel) će se izvršavati, jer greške u kernel prostoru mogu biti ozbiljnije.

---

11. Paralelno procesiranje

Paralelno procesiranje podrazumeva simultano izvršavanje zadataka kako bi se poboljšale performanse računarskih sistema.

U situacijama kada je datoteku od 1 GB moguće kopirati u RAM od 2 GB, kopiranje velikih datoteka može postati problematično sa većim veličinama, poput 10 GB.

Klasično rešenje za to je deljenje datoteke na manje delove, što smanjuje potrošnju memorije i omogućava procesoru da obavlja druge zadatke dok se čeka na kopiranje.

Ovaj pristup zahteva multitasking operativne sisteme koji podržavaju izvršavanje više simultanih zadataka.

Kada su dostupni višestruki procesori ili jezgra, moguće je dodatno poboljšati performanse paralelizacijom zadataka, uz saradnju među njihovim delovima.

Paralelne mašine se mogu grubo klasifikovati u višeprocesorske mašine i klastere, pri čemu se saradnja može ostvariti putem dijeljene memorije ili komunikacije porukama.

https://www.youtube.com/watch?v=3sGHTtYoW-Y

Unutar višeprocesorskih mašina, paralelizam se može postići na različitim nivoima, uključujući nivo instrukcija, Hyper-Threading, višestruka jezgra ili više CPU-a.

Izazov leži u upravljanju deljenim resursima, kao što je radna memorija, koja je sporija od procesora.

Problemi kao što je NUMA (Non-Uniform Memory Access) zahtevaju podršku operativnog sistema za efikasno raspoređivanje zadataka i alokaciju memorije.

Klasteri su povezani skupovi računara, pri čemu svaki čvor ima svoj operativni sistem i resurse.

Klasteri se koriste za poboljšanje stabilnosti i performansi, a saradnja putem dijeljene memorije obično se realizuje korišćenjem više niti, često putem OpenMP standarda.

Za komunikaciju između čvorova klastera koristi se MPI (Message Passing Interface).

Hibridni pristup, koji kombinuje OpenMP unutar čvorova i MPI između njih, omogućava efikasno korišćenje resursa. Kako su klasteri često skupi i korisnici žele da ih koriste, potrebni su sistemi za upravljanje poslovima (JMS(Job Managament Systems)) koji efikasno upravljaju resursima i raspoređuju zadatke.

Neki od popularnih sistema uključuju Condor, PBS, Torque i Maui, koji pomažu u efikasnom upravljanju klasterima.

---

12. Vrste klastera

Klasteri se klasifikuju prema svojim funkcijama i ciljevima, a najistaknutije vrste uključuju failover klastere, load-balancing klastere i high-performance klastere.

Failover klasteri povećavaju stabilnost sistema korišćenjem više čvorova koji obavljaju istu funkciju. U najjednostavnijoj formi, sastavljeni su od dva identična čvora; ako jedan otkaže, drugi preuzima funkciju. Međutim, izazov nastaje kada oba čvora deluju pogrešno. Da bi se ovo izbeglo, koristi se neparan broj čvorova kako bi se postigla kvalifikovana većina (kvorum), omogućavajući pravilno donošenje odluka.

Load-balancing klasteri raspodeljuju opterećenje između čvorova radi poboljšanja performansi. Ovi klasteri mogu sadržavati čvorove različitih performansi, što omogućava fleksibilnost u obradi zahteva. U internet servisima, ovakav pristup koristi HTTP protokol koji je pogodan za raspodelu opterećenja.

Raspodela se može vršiti jednostavnim DNS round-robin metodama ili putem specijalizovanih balansera opterećenja, koji nadziru rad servera i optimizuju raspodelu upita.

Međutim, pad balansera može dovesti do nedostupnosti svih servera, pa se preporučuje failover konfiguracija balansera.

High-performance klasteri su specijalizovani za pružanje visokih performansi, često po cenu specifičnih organizacija i tehnologija. Ova klasa se deli na tri podklase: **custom-made klasteri**, **Beowulf klasteri** i **COW (Cluster of Workstations)** klasteri.

- ***Custom-made klasteri*** su prilagođeni specifičnim potrebama korisnika i često koriste jedinstvene komponente, kao što su akceleratori za simulacije ili rudarenje kriptovaluta.

- ***Beowulf klasteri*** se sastoje od standardnih računara u minimalnim konfiguracijama, bez dodatnih perifernih uređaja, što smanjuje troškove i potrošnju energije.

- ***COW(Cluster of Workstations) klasteri*** koriste pune radne stanice koje se mogu koristiti i van klastera, a često su privremeni.

Mrežno povezivanje u HPC klasterima teži bržim vezama (bandwidth) i manjem kašnjenju (latency), često koristeći 40G Ethernet ili Infiniband, s tim da je Infiniband dominantna tehnologija. Topologija mreže može varirati od jednostavnih zvjezdastih do kompleksnijih konfiguracija poput grida ili hiperkubea, zavisno od svrhe klastera.

Važno je napomenuti da ne postoje jasne granice između ovih klasa klastera.

Dok se Beowulf i COW klasteri razlikuju u detaljima, load-balancing klasteri imaju slične ciljeve kao high-performance klasteri u poboljšanju performansi.

Međutim, HPC klasteri koriste napredne mrežne tehnologije za rešavanje složenih problema, dok su load-balancing klasteri često pogodniji za jednostavne ili slabo povezane probleme.

Ova fleksibilnost omogućava load-balancing klasterima da funkcionišu i kao failover klasteri kada imaju više resursa nego što je potrebno.

---

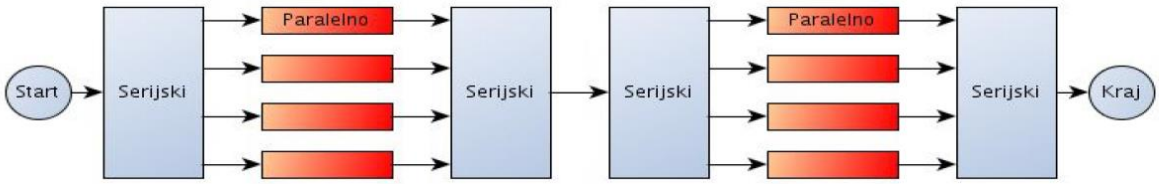
13. OpenMP

OpenMP (Open Multi-Processing) je standardizovani API koji olakšava rad sa nitima u programiranju.

Omogućava automatsko kreiranje i terminaciju niti, raspodelu opterećenja i sinhronizaciju, čime se pojednostavljuje paralelizacija kodova.

Ključna karakteristika OpenMP-a je njegova upotreba isključivo na mašinama sa deljenom memorijom, što znači da ne podržava distribuirane sisteme.

U osnovi, serijski kod se izvršava klasično, dok se određeni delovi koda paralelizuju, omogućavajući istovremeno izvršavanje više niti.



Programi u C i C++ koriste direktive kao što je `#pragma omp` za označavanje paralelnih sekcija. Na primer, uz jednostavnu direktivu, program može da se izvrši u više niti, što se može kontrolisati promenljivom `OMP_NUM_THREADS`. Programer takođe može da definiše broj niti unutar samog koda.

```
#include <omp.h>
#include <stdio.h>

int main (int argc, char **argv) {
  #pragma omp parallel
  {
     printf ("Moj prvi OpenMP program!\n");
  }
}
```

OpenMP omogućava efikasno sumiranje elemenata matrice kroz paralelizaciju.

U ovom procesu, promenljive se definišu kao privatne za svaku nit, a parcijalne sume se kombinuju u finalnu vrednost.

Raspodela iteracija može biti statička ili dinamička, što dodatno poboljšava efikasnost.

OpenMP takođe nudi različite strategije raspodele, kao što su statička, dinamička ili vođena raspodela, koje se mogu definisati iz komandne linije ili direktno u kodu.

Sama strategija raspodjele iteracija se može definisati na sljedeći način:

#pragma omp parallel for reduction(+:suma) schedule(strategija)

· **schedule(static, chunk-size)** - statički raspoređivač, dijeli petlju na dijelove veličine do chunk-size (default je približno iteracija/niti) i unaprijed raspoređuje posao po nitima sukcesivno

· **schedule(dynamic, chunk-size)** - dinamički raspoređivač, dijeli petlju (default chunk-size je 1!) i dinamički (i stohastički) dodjeljuje posao niti koja je trenutno slobodna

· **schedule(guided, chunk-size)** - dinamički raspoređivač, dijeli preostale iteracije po broju niti, chunk-size je minimalna preporučena veličina završnih iteracija

· **schedule(auto, chunk-size)** - kompajler bira najbolji

· **schedule(runtime, chunk-size)** - runtime definiše

Ovaj pristup čini OpenMP snažnim alatom za optimizaciju performansi aplikacija koje zahtevaju paralelno procesuiranje, a njegova jednostavna sintaksa olakšava integraciju u postojeće C i C++ projekte.

---

14. Message Passing Interface- MPI

Message Passing Interface (MPI) je ključna specifikacija za razmenu poruka u distribuiranim sistemima, gde deljena memorija nije moguća.

MPI se najčešće koristi u visokoperformantnom računarstvu i omogućava efikasnu komunikaciju između više nezavisnih mašina.

Ovo nije konkretna implementacija, već standardizovana biblioteka koja nudi alate za slanje i primanje poruka, kao i dodatne funkcije za upravljanje izvršavanjem programa.

MPI je podržan u raznim programskim jezicima, kao što su C, C++, Fortran i Java, i može se koristiti na različitim operativnim sistemima, uključujući Linux i Windows.

Osnovne funkcionalnosti MPI uključuju blokirajuću i neblokirajuću komunikaciju, kao i različite strategije za slanje poruka, što omogućava fleksibilno rešavanje različitih problema.

Unutar MPI biblioteke, postoje brojne funkcije, ali samo nekoliko osnovnih je ključno za većinu aplikacija: inicijalizacija, dobijanje broja procesa, slanje i primanje poruka, te završetak izvršavanja.

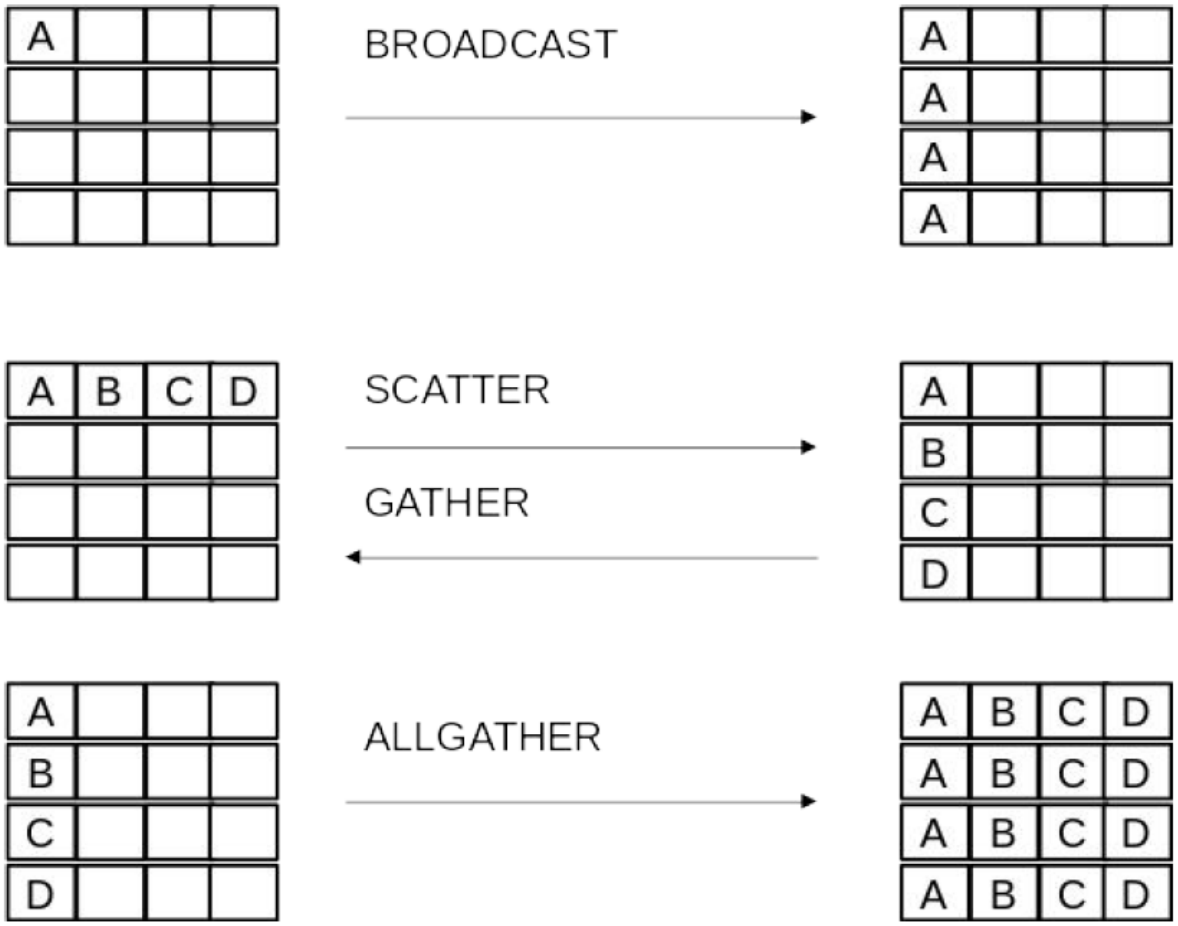
Ove funkcije omogućavaju jednostavno upravljanje procesima, a dodatna funkcionalnost može se koristiti za kompleksnije komunikacione obrasce.

Da bi se izvršili MPI programi, koristi se poseban kompajler (mpicc) i alat za pokretanje (mpirun), koji automatski upravlja pokretanjem procesa na različitim računarima.

MPI omogućava heterogeno izvršavanje, što znači da programi mogu raditi na različitim operativnim sistemima i arhitekturama bez problema.

Ovo čini MPI veoma pogodnim za rad u velikim klasterima i kompleksnim simulacijama, gde je efikasna komunikacija među procesima od ključnog značaja.

Primena MPI-ja obuhvata kolektivne komunikacije, kao što su slanje podataka sa jednog čvora na sve čvorove (broadcast), raspodela i prikupljanje podataka, što dodatno olakšava sinhronizaciju i razmenu informacija u distribuiranim sistemima.



---

15. Osnove UNIX filozofije

U doba kada je UNIX operativni sistem nastao, resursi računarskih sistema su bili značajno ograničeni.

Ovo je dovelo do fokusiranja na maksimalno iskorišćenje tih resursa, gde je upotreba asemblera bila skoro neizbežna, a programi su često bili kompleksni i monolitni.

Iako su takvi pristupi omogućili postizanje dobrih performansi, ubrzo se pokazalo da ne skaliraju dobro, proizvodeći složen kod koji je teško održavati i ponovno koristiti.

UNIX filozofija se temelji na razvoju softvera koji je minimalistički i modularan. Cilj je pisanje jednostavnog, jasnog i kratkog koda koji je lako testirati, izvršavati i održavati.

Ova ideja se ogleda u savremenim pristupima kao što je mikroservisna arhitektura, koja se fokusira na slabo povezane servise umesto velikih monolitnih sistema.

Dennis Ritchie i Ken Thompson su postavili nekoliko osnovnih principa tokom razvoja UNIX-a.

Prvo, kod treba da bude razumljiv i jednostavan za pisanje i održavanje. Drugo, interaktivnost je važna; sistemi dizajnirani za interakciju su lakši za prilagođavanje.

Takođe, važno je razvijati efikasan kod koji koristi resurse optimalno, a proces razvoja bi trebao biti vođen principom da se sistem razvija unutar samog UNIX-a, kako bi se prepoznali potencijalni problemi.

Doug McIlroy je dodao svoj doprinos UNIX filozofiji kroz principe kao što su "Pišite programe koji rade jednu stvar i rade je dobro" (DOTADIW) i "Očekujte da izlaz jednog programa može biti ulaz drugog". Ove ideje podstiču modularnost i omogućavaju lakšu kombinaciju alata.

Takođe, naglašava važnost ranog testiranja softvera i upotrebu automatizacije.

Konačno, ključni elementi UNIX filozofije su: usmeravanje na to da se obavi samo ono što je potrebno i sposobnost kombinovanja alata kako bi se postigla veća efikasnost.

Uvek treba raditi pametno, koristiti prethodno stečeno znanje i razmišljati o razlozima iza svake odluke u razvoju, umesto slepog praćenja trendova.

---

16. UNIX i rad sa periferijama

U upravljanju periferijskim uređajima i komunikaciji s njima, postoji niz pristupa koji se kreću od osnovnih do sofisticiranijih tehnika.

Kada su resursi u pitanju, najniži nivo uključuje pristupe kao što su direktno mapiranje memorije, prekidi i direktan pristup memoriji od strane periferija.

Jednom kada su ovi problemi rešenji na nivou operativnog sistema i drajvera, postavlja se pitanje kako omogućiti korisnicima da koriste te funkcionalnosti.

Za ovaj cilj često se definišu protokoli za pristup periferijskim uređajima, najčešće kroz neki API.

Unatoč očiglednim razlikama između različitih periferijskih uređaja, oni se mogu posmatrati sličnima kada su u pitanju njihova osnovna svojstva.

Na primer, tastatura i mikrofon su ulazni uređaji koji šalju podatke u vremenskim intervalima, dok se sekundarna memorija i grafička kartica mogu pristupiti u blokovima.

Ova sličnost može se povezati sa UNIX filozofijom, koja naglašava stvaranje API-ja koji omogućava rad s periferijskim uređajima bez potrebe za promenama u pristupu.

Kao što su periferijski uređaji često predstavljeni kao posebni blokovi memorije, moglo bi se primetiti da se sve može posmatrati kao fajl.

"Sve je fajl" filozofija

Ova filozofija se svodi na to da su svi uređaji i resursi predstavljeni kao fajlovi unutar operativnog sistema.

Ovo olakšava rad programerima jer ne moraju da uče različite načine pristupa različitim uređajima.

Umesto toga, sve se svodi na pristup specifičnim "fajlovima", a operativni sistem obavlja sve neophodne zadatke.

Primeri pristupa periferijama u Linux-u

1. **Serijski portovi**: Pristupaju se kroz fajlove kao što su `/dev/ttyS0`, `/dev/ttyS1`, itd.

2. **Diskovi**: Svi diskovi se nalaze u `/dev/` direktorijumu. Na primer, `/dev/sda` predstavlja prvi SCSI(Small Computer System Interface) ili SATA(Serial Advanced Technology Attachment) disk, dok bi `/dev/vdb` bio drugi disk unutar virtuelnog kontrolera.

SSD disk može biti predstavljen kao `/dev/nvme0n1`, što označava prvi NVMe interfejs.

3. **Memorija**: Fizička memorija može se pristupiti putem fajlova `/dev/mem` za svu fizičku memoriju, ili `/dev/kmem` za virtuelni adresni prostor kernela.

Takođe, možete menjati sadržaj određenih lokacija u fizičkoj memoriji jednostavno otvaranjem i pisanjem u odgovarajući fajl.

4. **Specijalni fajlovi**:

- **/dev/urandom**: Generiše slučajne brojeve.
- **/dev/zero**: Proizvodi beskonačan broj nula.
- **/dev/null**: "Crna rupa" za podatke, gde se podaci mogu slati, ali se ne čuvaju.

Ova jedinstvena filozofija "sve je fajl" omogućava pojednostavljenje komunikacije sa različitim uređajima,

pružajući konzistentan pristup koji je u skladu sa UNIX-ovim principima modularnosti i jednostavnosti.

Na taj način, programeri mogu efikasnije raditi sa periferijskim uređajima bez potrebe za kompleksnim podešavanjima ili razumevanjem različitih protokola za komunikaciju.

---

17. Linux Filesystem Hierarchy Standard (FSH)

Linux je jezgro operativnog sistema otvorenog koda i sam po sebi ne nameće bilo kakva ograničenja po pitanju sadržaja ili organizacije fajl sistema.

Iako se teoretski može imati sistem samo sa kernelom, praktično je smislenije imati operativni sistem koji pokreće init proces.

Ovaj pristup se koristi u ugrađenim sistemima, dok se funkcionalnost Linux-a često deli u različite distribucije koje kombinuju kernel sa sistemskim i korisničkim alatima.

Kako bi se olakšala organizacija fajl sistema, Linux fondacija je uspostavila standardnu hijerarhiju koja uključuje ključne direktorijume poput `/`, `/boot`, `/bin`, i `/etc`, kao i korisničke direktorijume kao što su `/home` i `/root`.

Od interesa za korisnike i servise namijenjene za upotrebu od strane korisnika su:

· /home korisnički direktorijumi (sadrži korisničke naloge npr. /home/student)

· /root korisnički direktorijum korisnika root (olakšava rad i otklanjanje problema,

jer /home nije neophodan za funkcionisanje sistema, a korisnik root gotovo

uvijek postoji u nekoj formi)

· /tmp privremeni fajlovi (ne mora biti na disku)

· /usr sekundarna hijerarhija read-only za korisničke programe (u idealnom slučaju)

sadrži bin, sbin, lib, itd. direktorijume

· /var promjenjivi fajlovi (podaci sistemskih aplikacija, logovi, ...), na primjer, često

se fajlovi koji čine web prezentaciju koju servira web server nalaze u

/var/www, a fajlovi koji predstavljaju MySQL bazu podataka u /var/lib/mysqlSve ove informacije upravljaju virtuelni fajl sistemi, koji omogućavaju pristup internim atributima operativnog sistema.

· /srv prostor predviđen za fajlove koji koriste serveri (web, FTP, itd)

često su u praksi u /var direktorijumu, kao što je ranije navedeno

· /opt korisnički programi - samodovoljni paketi koji sadrže sve što im treba,

uključujući biblioteke i pomoćne programe koji bi inače bili instalirani na

nivou sistema i raspoređeni u druge direktorijume, ali često kod distribuiranja

komercijalnog softvera to nije izvodivo rješenje pa se koristi /opt (takođe

olakšava uklanjanje softvera jednostavnim brisanjem direktorijuma)

Direktorijumi posebne namjene su:

· /dev uređaji (devices)

· /media uklonjivi/privremeni uređaji (DVD, USB memorija, itd)

· /mnt dugoročno priključeni fajl sistemi (mrežni, diskovi, itd)

· /run trenutne informacije o sistemu

· /sys informacije o uređajima, drajverima, itd.

· /proc informacije o procesima, kernelu, itd.

Ova organizacija omogućava bolju efikasnost i jednostavnije upravljanje resursima, reflektujući osnovne UNIX ideje koje su i dalje prisutne u modernim operativnim sistemima.

---

18. Osnove kontrole pristupa

Kontrola pristupa u računarskim sistemima obuhvata autentifikaciju, autorizaciju i provođenje politika pristupa.

Autentifikacija se koristi za potvrđivanje identiteta korisnika i može se vršiti putem tri vrste faktora: znanja (kao što su šifre), posjedovanja (poput ID kartica) i biometrijskih podataka (npr. otisci prstiju). Upotreba više faktora iz različitih kategorija, poznata kao višefaktorska autentifikacija, povećava sigurnost u odnosu na jednofaktorsku autentifikaciju.

Međutim, klasični sistemi autentifikacije često imaju nedostatke jer se autentifikacija obavlja samo pri prijavi, a ukoliko napadač dobije pristup uređaju, zaštita ne pomaže.

Kontinuirana autentifikacija nudi rešenje praćenjem korisničkih aktivnosti kako bi se detektovao sumnjiv pristup ili neuobičajeno ponašanje.

Autorizacija definiše politike pristupa, dok provođenje ovih politika obezbeđuje da sistem ne dospe u neautorizovano stanje. Problemi nastaju kada se pokušava omogućiti pristup fajlovima bez adekvatne kontrole, što može dovesti do sigurnosnih propusta.

Pravilna implementacija ovih kontrola je ključna za očuvanje sigurnosti sistema.

---

19. Modeli kontrole pristupa

Postoji mnogo načina za definiranje i provođenje politika pristupa, a ključno je utvrditi ko ima pravo da ih postavlja i za koje objekte.

Najčešće se koristi primjera fajl sistema, ali s porastom društvenih mreža, ovaj problem je dobio nove dimenzije, uključujući pitanja privatnosti.

Četiri osnovna modela kontrole pristupa su:

1. **Discretionary Access Control (DAC)**: Korisnik ima diskreciona prava na objekte koje posjeduje, a sistem ne intervenira u proces.

2. **Mandatory Access Control (MAC)**: Sistem propisuje i provodi pravila pristupa, obično se koristi u visoko zaštićenim okruženjima.

3. **Role-Based Access Control (RBAC)**: Prava pristupa zavise od uloge korisnika u sistemu, može kombinovati DAC i MAC.

4. **Attribute-Based Access Control (ABAC)**: Nudi visoku fleksibilnost, ali može biti kompleksan za implementaciju.

Svaki model ima svoje prednosti i nedostatke, a inženjeri moraju birati rješenja koja su optimalna za specifične probleme.

Pravilnim odabirom pristupa osigurava se ne samo trenutna efikasnost, već i budući razvoj sistema, što se često povezuje s modularnim rješenjima inspirisanim UNIX filozofijom.

---

20. Modeli kontrole pristupa- DAC

U DAC (Discretionary Access Control) modelu kontrole pristupa, korisnik (vlasnik objekta) ima potpunu slobodu upravljanja politikom pristupa tom objektu, bez sistemskih ograničenja. Postoje dva pristupa DAC-u:

1. **Owner based**: Vlasnik može slobodno definisati politiku pristupa i prenositi vlasništvo na druge korisnike.

2. **Capabilities based**: Korisnici ne moraju biti identifikovani kao vlasnici, već im je dovoljno da posjeduju ključ (ili token) za pristup objektu, koji mogu dijeliti s drugima.

DAC se često koristi u višekorisničkim operativnim sistemima i fajl sistemima (npr. ext2, NTFS), gdje korisnik koji kreira fajl postaje njegov vlasnik s punim pravima pristupa. U savremenim internet sistemima, korisnici mogu dijeliti dokumente s drugima putem linkova, omogućavajući pristup bez prijave, što ilustrira capabilities pristup.

Također, DAC model se primjenjuje i u dijeljenju memorije između procesa u operativnom sistemu, gdje procesi sami odlučuju o dijeljenju.

Različite kompleksnosti provjera prava pristupa između memorije i internet resursa igraju važnu ulogu u performansama sistema.

---

21. Modeli kontrole pristupa- MAC

U MAC (Mandatory Access Control) modelu kontrole pristupa, sistem određuje obavezna prava pristupa, a korisnicima i vlasnicima se ne ostavlja mnogo slobode u definisanju politika.

MAC modeli su prvotno razvijeni za vojne i osjetljive sisteme, gdje je sigurnost ključna, te se često smatraju "zlatnim standardom".

Ovaj model se koristi i u civilnim sektorima, kao što su zdravstvo i finansije, gdje su strogo propisani pristupi podacima i transakcijama.

U MAC sistemima se često koristi koncept višenivovske sigurnosti (MLS), koji omogućava kreiranje hijerarhije prava pristupa putem oznaka.

Primjeri operativnih sistema koji implementiraju MAC uključuju Microsoft Windows Server 2008, FreeBSD i Linux, koji omogućava razvoj sigurnosnih proširenja putem LSM (Linux Security Modules) interfejsa.

Unutar Linux-a, postoje različita rešenja kao što su SELinux, AppArmor i Smack, koja omogućavaju kontrolu resursa i podržavaju MAC model na različite načine, uključujući i ugrađene sisteme.

---

22. Modeli kontrole pristupa – RBAC

Kada DAC i MAC modeli ne zadovoljavaju potrebe sistema, često se koristi RBAC (Role-based Access Control) model kontrole pristupa.

Ovaj model, neutralan po pitanju politike pristupa, može emulirati i MAC i DAC, ali nudi i dodatnu fleksibilnost.

RBAC se oslanja na uloge, prava pristupa i korisnike, a prava nisu ograničena samo na pristup objektima, već se mogu preciznije definirati (npr. dozvole za dodavanje ili inkrementovanje podataka).

RBAC se temelji na tri pravila: korisnik može koristiti prava samo ako mu je dodijeljena uloga, trenutna uloga mora biti autorizovana, a prava moraju biti autorizovana za tu ulogu.

Ovo pojednostavljuje administraciju, posebno u sistemima s velikim brojem korisnika i prava.

Hijerarhijski RBAC sistemi dodatno olakšavaju upravljanje pravima, omogućavajući nasljeđivanje prava između uloga (npr. nastavnik nasljeđuje prava asistenta i demonstratora).

Ovaj pristup smanjuje mogućnost grešaka i pojednostavljuje buduće održavanje politika.

RBAC se često koristi u višekorisničkim web aplikacijama, a većina razvojnih okvira pruža podršku za ovaj model.

---

23. Modeli kontrole pristupa – ABAC

ABAC (Attribute-based Access Control), poznat i kao PBAC (Policy-based Access Control) ili CBAC (Claims-based Access Control) u Microsoft okruženju, nudi visoku fleksibilnost u definisanju politika pristupa putem kompleksnih logičkih izraza.

Model se temelji na atributima koji se dijele u četiri osnovne grupe:

1. **Atributi subjekta** - opisuju korisnika (npr. korisničko ime, IP adresa).

2. **Atributi akcije** - opisuju namjeravanu akciju (npr. čitanje, pisanje).

3. **Atributi objekta** - opisuju objekat nad kojim se akcija izvršava (npr. status dokumenta).

4. **Atributi konteksta** - opisuju okolnosti izvršenja akcije (npr. trenutni datum i vrijeme).

Fleksibilnost ABAC-a omogućava precizno definisanje prava pristupa u zavisnosti od različitih uslova, poput ograničavanja pristupa fajlovima na osnovu radnog vremena ili lokacije.

Arhitektura ABAC sistema uključuje:

- **PEP (Policy Enforcement Point)**: Tačka koja štiti pristup objektima analizirajući zahtjeve.

- **PDP (Policy Decision Point)**: Tačka koja donosi odluke o odobravanju ili odbijanju pristupa na osnovu definisanih politika.

- **PIP (Policy Information Point)**: Omogućava povezivanje PDP-a s vanjskim izvorima podataka za dodatne informacije.

Za razliku od DAC, MAC i RBAC modela, ABAC može koristiti dodatne izvore informacija, što povećava njegovu sposobnost.

Iako mnogi RBAC sistemi podržavaju funkcionalnosti slične ABAC-u, ABAC se ističe svojom složenijom i fleksibilnijom logikom upravljanja pristupom.

24. Osnove fajl sistema – fajlovi

Imenovanje Fajlova

Fajlovi su osnovni mehanizam za skladištenje i pristup informacijama na disku, a njihov naziv, struktura i operacije zavise od fajl sistema. Imenovanje fajlova omogućava identifikaciju čak i nakon što proces koji ih kreira završi. Pravila za imenovanje variraju; mnogi sistemi podržavaju nazive do 255 karaktera, uključujući brojeve i specijalne znakove. Ekstenzije fajlova često opisuju njihovu svrhu (npr. .c, .html), a neki sistemi, poput Windows-a, vezuju ekstenzije za određene programe, dok ih UNIX tretira kao konvenciju.

Struktura Fajlova

Struktura fajlova može biti nestrukturisana (niz bajtova), strukturisana kao sekvenca zapisa fiksne dužine, ili organizovana kao stablo za efikasnu pretragu. Većina savremenih operativnih sistema koristi nestrukturisani pristup zbog fleksibilnosti.

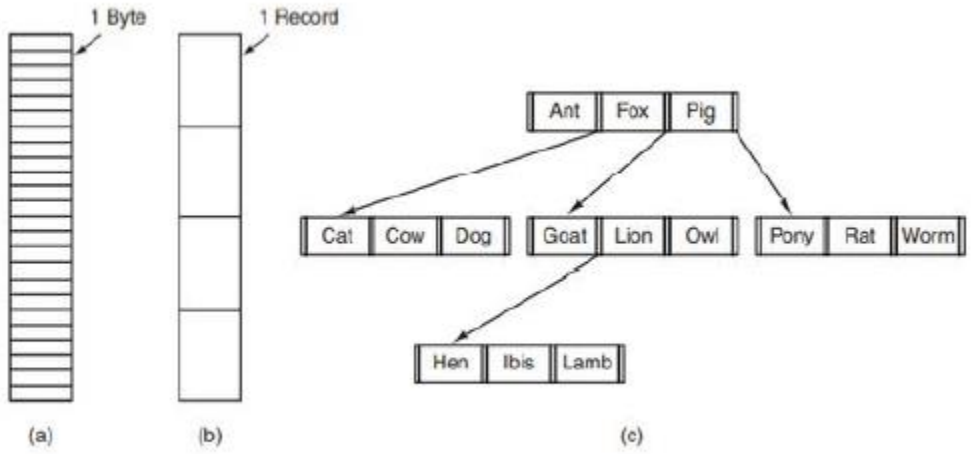


Figure 4-2. Three kinds of files. (a) Byte sequence. (b) Record sequence. (c) Tree

Tipovi Fajlova

Fajlovi mogu biti regularni (tekstualni ili binarni), direktorijumi ili specijalni fajlovi za ulaz/izlaz. Regularni fajlovi služe za čuvanje korisničkih podataka, dok binarni fajlovi često imaju specifičnu internu strukturu.

Pristup Fajlovima

Operacije sa fajlovima uključuju kreiranje, brisanje, otvaranje, zatvaranje, čitanje, pisanje, dodavanje (append), i promenu pozicije (seek). Atributi fajlova, poput datuma modifikacije, veličine i pristupnih prava, omogućavaju dodatne funkcionalnosti poput organizacije, sigurnosti i efikasnosti. Random pristup je ključan za moderne aplikacije, dok su sekvencijalni pristupi više istorijskog značaja.

25. Osnove fajl sistema – direktorijumi

Kako bi vodili računa o fajlovima, fajl sistemi obično imaju direktorijume ili foldere, koji su takođe fajlovi.

Jednodelni direktorijumski system (Single Level Directory System)

- U ovom sistemu postoji samo jedan direktorijum koji sadrži sve fajlove.
- Prednost: jednostavnost i brza pretraga fajlova.
- Uglavnom se koristi u jednostavnim uređajima poput digitalnih kamera i nekih muzičkih plejera.
- Primer: CDC 6600, prvi svetski superračunar, imao je sličan sistem zbog jednostavnosti dizajna.

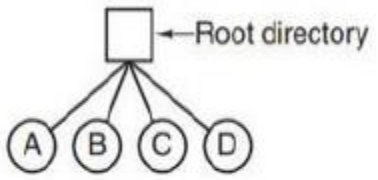


Figure 4-6. A single-level directory system containing four files.

Hijerarhijski direktorijumski sistem

- Koristi se za organizaciju velikog broja fajlova, gde su fajlovi grupisani u poddirektorijume.
- Svaki korisnik može imati svoj privatni koreni direktorijum, čime se olakšava organizacija rada.
- Na primer, korisnici mogu kreirati poddirektorijume za projekte na kojima rade.
- Ovaj sistem je standard u modernim operativnim sistemima.

Hierarchical Directory Systems

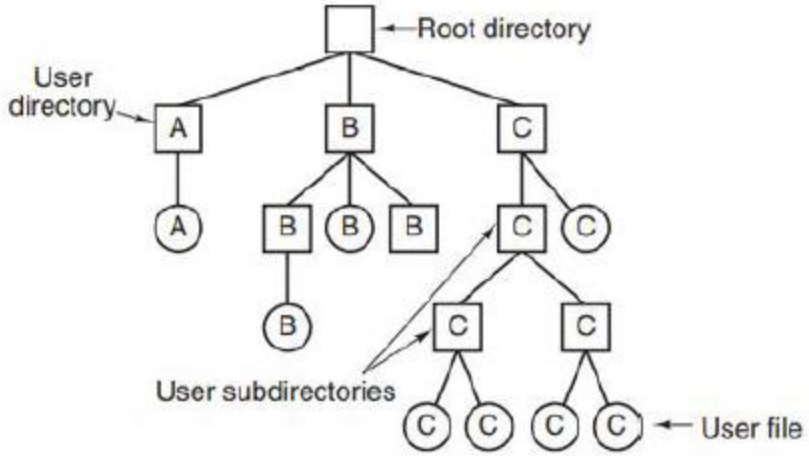


Figure 4-7. A hierarchical directory system.

Nazivi putanja

- **Apsolutne putanje**: Počinju od korenog direktorijuma i sadrže celu putanju do fajla. Na primer:
  - UNIX: `/usr/ast/mailbox`
  - Windows: `\usr\ast\mailbox`
  - MULTICS: `>usr>ast>mailbox`
- **Relativne putanje**: Odnose se na trenutni radni direktorijum. Na primer, ako je trenutni direktorijum `/usr/ast`, fajl se može referencirati kao `mailbox`.

Operacije s direktorijumima

1. **Kreiranje** (Create): Kreiranje novog, praznog direktorijuma.
2. **Brisanje** (Delete): Moguće je obrisati samo prazan direktorijum.
3. **Otvori direktorijum** (Opendir): Da bi se čitali fajlovi unutar direktorijuma, direktorijum mora biti otvoren.
4. **Zatvori direktorijum** (Closedir): Zatvaranje direktorijuma nakon čitanja.
5. **Pročitaj direktorijum** (Readdir): Vraća sledeći unos u otvorenom direktorijumu.
6. **Preimenuj** (Rename): Mogućnost preimenovanja direktorijuma.
7. **Uvezivanje** (Link): Povezivanje fajla u više direktorijuma (hard-link).

8. **Uklanjanje** (Unlink): Uklanja unos iz direktorijuma, a može i da izbriše fajl iz sistema.

Simboličko uvezivanje
- **Simbolički link**: Kreira link koji pokazuje na drugi fajl, omogućavajući premošćavanje granica između diskova, ali je manje efikasan od hard-linka.

Ova struktura i operacije omogućavaju korisnicima efikasno upravljanje fajlovima i direktorijumima, posebno u složenijim sistemima sa više korisnika i većim brojem fajlova.

---

26. Žurnalski fajl sistemi

Žurnalski fajl sistemi su dizajnirani da poboljšaju pouzdanost i brzinu operacija u slučaju pada sistema. Ključna ideja je da se prvo beleže planirane operacije pre nego što se izvrše, što omogućava da se u slučaju kvara sistem vrati u konzistentno stanje.

Kako funkcionišu žurnalski fajl sistemi?
1. **Zapisivanje operacija**: Pre nego što se izvrše operacije kao što su brisanje fajla, sistem prvo kreira zapis (log entry) o tim operacijama.

2. **Pisanje na disk**: Ovaj zapis se upisuje na disk, a može se ponovo pročitati radi verifikacije.

3. **Izvršavanje operacija**: Nakon što je zapis uspešno upisan, operacije se izvršavaju.

4. **Brisanje zapisa**: Kada se operacije završe, zapis se može obrisati.

U slučaju pada sistema:

- Nakon ponovnog pokretanja, sistem proverava zapise u žurnalu da vidi da li su neke operacije ostale neizvršene.

- Ako jesu, te operacije se ponovo pokreću dok se ne izvrše ispravno.

Idempotentne operacije

Operacije moraju biti dizajnirane tako da su idempotentne, što znači da se mogu ponavljati bez dodatnog efekta. ***Ovo je ključno za oporavak, jer omogućava da se iste operacije izvrše više puta bez problema***.

Primeri idempotentnih operacija:

- Oslobađanje blokova ili i-čvorova (ako su već oslobođeni, nema problema).

- Pretraga direktorijuma i uklanjanje unosa po imenu.

Primeri ne-idempotentnih operacija:

- Dodavanje blokova na slobodnu listu bez prethodne provere da li su već prisutni.

Atomske transakcije

Žurnalski sistemi često implementiraju koncept atomskih transakcija, što znači da se operacije mogu grupisati tako da će ili sve biti izvršene ili nijedna.

Ovo dodatno poboljšava sigurnost i pouzdanost operacija.

Primena žurnalskih fajl sistema

- **NTFS**: Microsoft-ov žurnalski fajl sistem koji je u razvoju od 1993. godine. Poznat je po svojoj robusnosti i redosledu operacija koje minimiziraju rizik od gubitka podataka.

- **Linux ext3**: Manje ambiciozan od ReiserFS-a, ali je kompatibilan sa ext2, što olakšava migraciju i upotrebu.

- **ReiserFS**: Prvi Linux žurnalski fajl sistem, ali je izgubio popularnost zbog nedostatka kompatibilnosti s ext2.

Žurnalski fajl sistemi su posebno korisni u okruženjima gde je stabilnost i brzina oporavka kritična, kao što su serveri i sistemi sa velikim količinama podataka.

---

27. Virtuelni fajl sistem

Virtuelni fajl sistem (VFS) je složen arhitektonski koncept koji omogućava integraciju više različitih fajl sistema u jedinstvenu strukturu, što olakšava rad sa podacima bez potrebe da korisnici znaju o detaljima implementacije svakog pojedinačnog fajl sistema. Ovaj pristup se često koristi u modernim UNIX i Linux sistemima.

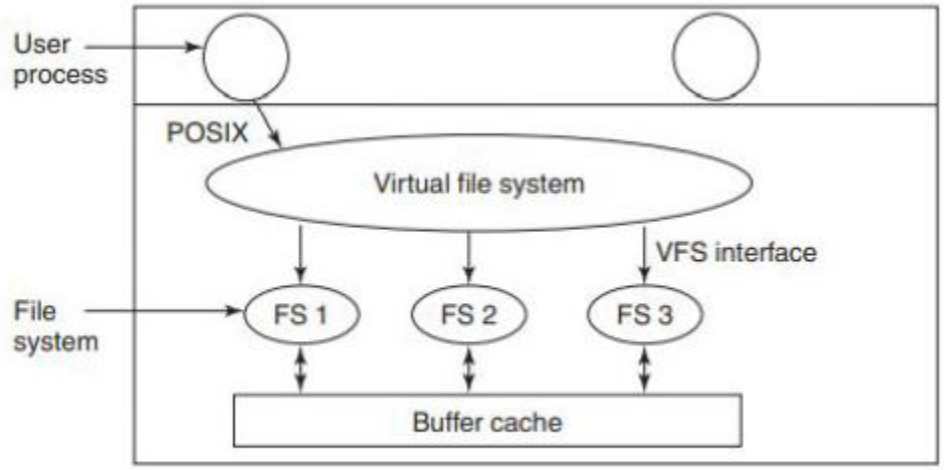


Figure 4-18. Position of the virtual file system.

Ključni aspekti VFS-a

1. **Integracija više fajl sistema**: U većini operativnih sistema, poput Windows-a, različiti fajl sistemi (npr. NTFS, FAT32) su odvojeni i identifikovani slovima drajvova (C:, D:, itd.). U UNIX sistemima, VFS omogućava integraciju različitih fajl sistema u jedinstvenu hijerarhiju, omogućavajući korisnicima da interaguju sa njima kao sa jednim celinom.

2. **Apstrakcija**: VFS uvodi apstraktni sloj koji definiše zajedničke operacije fajl sistema, kao što su otvaranje, čitanje i pisanje fajlova. Ova apstrakcija omogućava sistemskim pozivima da budu standardizovani (npr. POSIX interfejs), bez obzira na to koji konkretni fajl sistem se koristi u pozadini.

3. **Dva interfejsa**:

   - **Gornji interfejs**: Korisnički procesi komuniciraju sa VFS-om putem standardnih sistemskih poziva (open, read, write, itd.).

- **Donji interfejs**: VFS komunicira sa različitim fajl sistemima koristeći funkcijske pozive koji su specifični za te sisteme.

4. **Registracija fajl sistema**: Kada se sistem pokrene, koreni fajl sistem se registruje na VFS. Dodatni fajl sistemi se mogu "mount-ovati" tokom rada, što im omogućava da postanu dostupni u hijerarhiji.

Kako VFS funkcioniše?

1. **Pokretanje i registracija**:

- Kada se operativni sistem pokrene, registruje se koreni fajl sistem. Dodatni fajl sistemi se registruju kada se mount-uju, bilo tokom pokretanja ili dinamički kasnije.

2. **Obrada sistemskih poziva**:

- Kada korisnički proces želi da otvori fajl, šalje poziv VFS-u. VFS koristi putanju do fajla da locira odgovarajući fajl sistem, pronalazi njegov superblock, i koristi ga za pristup informacijama o fajlu.

3. **Kreiranje v-čvora**:

- VFS kreira strukturu podataka poznatu kao v-čvor (v-node) koja sadrži informacije o fajlu. Ova struktura se koristi za praćenje stanja fajla i operacija nad njim.

4. **Tabela fajl deskriptora**:

- Kada se fajl uspešno otvori, VFS pravi unos u tabeli fajl deskriptora koja prati sve otvorene fajlove, povezanu sa v-čvorom.

5. **Izvršavanje operacija**:

- Kada korisnički proces izvrši operaciju (npr. čitanje fajla), VFS locira v-čvor, a zatim poziva odgovarajuću funkciju konkretnog fajl sistema putem adrese u tabeli funkcija.

Prednosti VFS-a

- **Transparentnost**: Korisnici ne moraju brinuti o raznim fajl sistemima koji su u pozadini; svi se ponašaju kao jedinstvena cjelina.

- **Fleksibilnost**: Lako je dodati nove fajl sisteme bez potrebe za značajnim promenama u osnovnom kodu operativnog sistema.

- **Podrška za udaljene fajl sisteme**: VFS može raditi sa fajl sistemima koji se nalaze na mreži (npr. NFS), čime se omogućava pristup udaljenim podacima kao da su lokalni.

Zaključak

Virtuelni fajl sistem je ključna komponenta modernih UNIX i Linux operativnih sistema, omogućavajući efikasno upravljanje podacima i fleksibilnost u radu sa različitim fajl sistemima. Njegov dizajn olakšava rad programerima i korisnicima, pružajući jedinstven i dosledan interfejs za interakciju sa fajlovima i direktorijumima.

---

28. ZFS fajl sistem, integritet i RAIDZ

ZFS (Zettabyte File System) je kombinacija fajl sistema i menadžera volumena, prvobitno open-source, a sada licenciran Oracle-om, dok se OpenZFS održava kao besplatna verzija.

Ovaj sistem upravlja i fizičkim blokovima (kao što su diskovi) i logičkom memorijom, omogućavajući potpunu kontrolu nad skladištenjem podataka. ZFS je dizajniran da štiti podatke od grešaka i korupcije, garantujući da su svi podaci verifikovani i optimizovani.

Sadrži mehanizme za snepšotove i replikaciju, što omogućava restauraciju prethodnih stanja fajl sistema.

ZFS koristi hijerarhijsko kreiranje kontrolnih suma za sve podatke, automatske popravke i samozaceljivanje u slučaju grešaka.

Takođe, ugrađuje upravljanje RAID nivoima, kao i mehanizme za kompresiju i deduplikaciju podataka.

Kada dođe do greške u RAID uređaju, ZFS ostaje funkcionalan, jer upravlja i RAID-om.

Osobina integriteta podataka omogućava zaštitu od tihe korupcije kroz korišćenje Flečerovog algoritma i SHA heševa.

ZFS je 128-bitni sistem, podržava enkripciju i omogućava promenu ključeva tokom rada.

Osim što koristi različite nivoe keširanja, kao što su ARC i SLOG, ZFS se oslanja na RAID-Z (paritetni RAID 5) i disk mirroring (RAID 1) umesto tradicionalnog RAID-a, čime omogućava brže operacije i fleksibilnost.

Integracija fajl sistema i RAID-a omogućava bolju validaciju blokova i održavanje integriteta podataka.

---

29. Isporuka aplikacija i repozitorijumi paketa

Problem isporuke aplikacija proističe iz nedostatka kontrole nad okruženjem u kojem će se aplikacije izvoditi, što otežava garantovanje njihove funkcionalnosti na korisničkim računarima. ***Aplikacije se sastoje od koda, podataka, okruženja i procedura instalacije, a svaka od ovih komponenti može predstavljati izazov.***

Osnovne komponente uključuju izvršni kod i potrebne biblioteke, a postoji rizik da potrebni alati nisu instalirani ili da su pogrešne verzije.

Kako bi se olakšala isporuka, razvijaju se procedure instalacije koje osiguravaju da su sve zavisnosti ispunjene, ali te procedure mogu postati podjednako kompleksne kao i same aplikacije. U Linuxu, upravljanje paketima predstavlja popularan način isporuke softvera, gde paketi sadrže potrebne fajlove i podatke o zavisnostima.

Repozitorijumi paketa omogućavaju korisnicima jednostavno pronalaženje, instalaciju i ažuriranje softvera, uz podršku digitalnih potpisa koji osiguravaju poverenje.

Međutim, potreban je dodatni trud za podršku različitim distribucijama i verzijama, s obzirom na raznovrsnost platformi na kojima Linux funkcioniše. Time se otežava održavanje i osiguranje da aplikacije funkcionišu ispravno na različitim sistemima.

---

30. Isporuka samostalnih Linux aplikacija

Isporuka samostalnih Linux aplikacija može biti izazovna, posebno kada se ne želi koristiti paketski način rada ili kada aplikacije nisu deo zvaničnih repozitorija.

Glavni problem je obezbeđivanje ponovljivog izvršavanja aplikacije u različitim okruženjima.

Često se koristi pristup stvaranja izolovanog okruženja, poznatog kao "***sandbox***", kako bi se minimizirao uticaj na sistem.

Jedan od ranijih pristupa bio je ***CDE (Code + Data + Environment***), koji je omogućavao praćenje svih zavisnosti tokom izvršavanja, ali nije bio najefikasniji.

Danas se koristi metoda "slika aplikacije" (application image), koja pakira sve zavisne resurse i obezbeđuje veću sigurnost.

****AppImage**** je jedan od popularnih sistema koji omogućava jednostavno izvršavanje aplikacija iz jednog fajla, ali zahteva ručne instalacije i ažuriranja, što može uticati na korisničko iskustvo.

****Flatpak**,*** koji je popularan kod GUI aplikacija, zahteva dodatnu infrastrukturu i omogućava precizno podešavanje dozvola, što olakšava bezbedan rad.

Ima i repozitorijum (Flathub) koji poboljšava korisničko iskustvo automatizacijom instalacije i ažuriranja.

****Snap**** je sistem koji podržava širok spektar aplikacija, uključujući i sistemske servise, i koristi slojeve za efikasnije korišćenje resursa i ažuriranje.

Ovaj pristup omogućava deljenje osnovnih fajlova između aplikacija, smanjujući potrebu za velikim instalacijama.

Sve tri metode (AppImage, Flatpak, Snap) nude različite pristupe isporuci aplikacija, prilagođavajući se potrebama korisnika i zahtevima okruženja.

---

31. Virtuelizacija

Virtuelizacija nije nov pojam. Prva demonstracija je izvršena na računaru IBM S/360-67 još davne 1967. godine upotrebom CP-40, a godinu kasnije i upotrebom tadašnjeg "open source" rješenja pod nazivom CP/CMS. Sistem je omogućavao *time-sharing* izvršavanje više programa na jednom računaru (i imao hardversku podršku za virtuelnu memoriju).
Popek i Goldberg su 1974. godine postavili teorijske zahtjeve za efikasnu virtuelizaciju, koji iako su zasnovani na pojednostavljenom modelu i danas imaju bitno mjesto u oblasti virtuelizacije.

Prema njima **virtuelne mašine** (**VM**) su sposobne da virtuelizuju puni skup hardverskih resursa, uključujući procesor, memoriju (primarnu i sekundarnu) i periferije.
Ovakva virtuelizacija,
ostvarena od strane softvera pod nazivom **hipervizor** ili **Virtual Machine Monitor** (**VMM**) mora da zadovolji sljedeće uslove:

· **Ekvivalentnost**

◦ Program koji se izvršava u okviru VM treba da se ponaša na identičan način kao da se

izvršava na stvarnoj mašini.

· **Kontrola resursa**

◦ VMM mora u potpunosti kontrolisati sve virtuelizovane resurse.

· **Efikasnost**

◦ Statistički značajna većina instrukcija mora biti izvršena bez uplitanja VMM.

Danas je u upotrebi veliki broj različitih definicija virtuelne mašine, od kojih su tri najkraće:
izolacija aplikacije od hardvera;
stvaranje okruženja za emulaciju operativnog sistema;
efikasna,izolovana kopija realnog sistema.

Pri izvršavanju koda u VM možemo razlikovati dva osnovna tipa instrukcija: osjetljive (one koje mijenjaju stanje procesora na privilegovan način, npr. PUSHF i POPF) i neosjetljive (koje to ne rade, npr. JZ i NOP). VMM može da izvrši kod VM i da ostane u potpunoj kontroli resursa na dva osnovna načina. Prvi način podrazumijeva da postoji **hardverska podrška** u ISA koja omogućava da **kritične** instrukcije (osjetljive, a neprivilegovane) obradi procesor na efikasan način (ili da su sve osjetljive instrukcije privilegovane) i obično se naziva **hardverski potpomognuta virtuelizacija**. Drugi pristup se naziva **dinamičko rekompajliranje** i zasniva se na ideji da je ponašanje kritičnih instrukcija (ili većih dijelova koda) moguće reprodukovati zamjenom serijom nekritičnih instrukcija u toku izvršavanja. VMM praktično konstantno nadzire instrukcije koje treba da izvrši u budućnosti i, ako detektuje kritične instrukcije, mijenja ih sekvencom koja mu omogućava da zadrži punu kontrolu nad svim resursima na efikasan način.

Ovo je jedini način
virtuelizacije moguć na arhitekturama bez hardverske podrške za virtuelizaciju.
Ovaj princip se ne koristi samo za virtuelizaciju, npr. JMV i Valgrind koriste ovaj pristup.
Važna osobina VMM je mogućnost privremenog zaustavljanja VM i pravljenja slike svih
resursa (***snapshot***) što omogućava savršen bekap ili migraciju VM na drugi server (npr. Privremeno zaustavimo VM, iskopiramo njeno stanje na drugi server i ponovo je pokrenemo na njemu).

---

32. Vrste hipervizora

U klasičnim pregledima oblasti virtuelizacije, uključujući i Popeka i Goldberga, često je navedena podjela na tipove hipervizora (VMM-a).

U ovoj, prilično strogoj, podjeli, **hipervizori tipa 1** su oni koji se izvršavaju direktno na hardveru i služe samo da obezbijede virtuelnim mašinama okruženje za izvršavanje.
U modernim okvirima u ovaj tip se obično ubrajaju VMWare ESXi, Microsoft Hyper-V i Oracle VM Server.
S druge strane, **hipervizori tipa 2** su hipervizori koji se izvršavaju pod već postojećim operativnim sistemom, uporedo sa drugim aplikacijama i servisima. Moderni predstavnici su VirtualBox, VMWare Player, QEMU, itd.

Problem nastaje kod klasifikovanja hipervizora na Linux i FreeBSD platformama koji su realizovani kao kernel moduli koji praktično od kernela osnovnog operativnog sistema stvaraju
hipervizor (znači tip 1) ali je taj osnovni operativni sistem i dalje potpuno funkcionalan I omogućava korisniku da pokreće i druge aplikacije uporedo sa virtuelnim mašinama (znači tip 2).

Tip 1 se još naziva i *Standalone VMM* jer je sam odgovoran za sve. Izvršava se "ispod" svih operativnih sistema (koji su "gosti" u virtuelnim mašinama) i, samim tim, direktno pristupa hardveru.
To takođe znači da je za upotrebu bilo koje hardverske komponente potrebno imati i odgovarajuće drajvere za hipervizor.
Kako je tržište ovakvih hipervizora mnogo manje od klasičnog tržišta koje pokriva, npr. Windows operativni sistem, samim tim je i proporcionalno manja podrška proizvođača komponenti u vidu fabričkih drajvera. Tim slijedom dolazimo do zaključka da je u slučaju upotrebe hipervizora tipa 1 neophodno pažljivo provjeriti da li su sve komponente podržane, tj. da li postoji odgovarajući drajver za željenu verziju hipervizora.

Kako je ovo netrivijalan problem, hipervizori tipa 1 su gotovo isključivo serverska rješenja i obično proizvođači nude gotove osnovne konfiguracije već prilagođene određenim konkretnim proizvodima (npr. HP i Dell vam pri specifikaciji servera nude opcije za VMWare, Hyper-V ili Xen).

Hipervizori tipa 2 se još nazivaju i *Hosted* jer se izvršavaju u okviru postojećeg operativnog sistema koji tada ima ulogu "domaćina", odnosno hosta.
Kako sada hipervizor ne pristupa direktno hardveru nego koristi usluge osnovnog operativnog sistema, nema ni potrebe za postojanjem drajvera za pojedine komponente (jer je moguće koristiti ih kroz osnovni OS).
S jedne strane je to dobra osobina jer gotovo sve što možemo da kupimo ima drajvere za popularne operativne sisteme, a s druge strane je loša jer nemamo nikakvu garanciju da su ti drajveri dovoljno kvalitetni I pouzdani za ozbiljnu virtuelizaciju. Imajući navedeno u vidu, nije iznenađenje što se ovaj vid hipervizora obično koristi kao *desktop* bazirano rješenje i nije pretjerano popularno u serverkoj virtuelizaciji.

Treba voditi računa i o tome da je u ovom slučaju određeni dio resursa neminovno dodijeljen i osnovnom operativnom sistemu i te resurse ne možemo da koristimo za potrebe virtuelnih mašina tako da, ako je riječ o namjenskom virtuelizacijskom serveru, mi praktično kupujemo hardver koji neće biti iskorišten.

Pored potpune virtuelizacije, moguće je vršiti i ograničenu virtuelizaciju pri čemu se ne vrši emuliranje ponašanja kompletnog računarskog sistema. Ideja je da se operativni sistem u virtuelnoj mašini prilagodi izvršavanju u istoj tako što se vrši modifikacija dajvera ili i samog kernela čime dobijamo **paravirtuelizaciju**.
Logičan nastavak ovog procesa je potpuno prilagođavanje gostujućeg operativnog sistema eliminacijom njegovog kernela i upotrebom osnovnog (host) kernela za emulaciju postojanja virtuelnog računara - ovaj put samo na nivou kernela.

Ovaj pristup se naziva **virtuelizacija na nivou operativnog sistema**.

---

33. VMWare virtuelizacija

VMWare je kompanija poznata po rješenjima za virtuelizaciju, poput VMWare Workstation (desktop) i ESX/vSphere (serverska virtuelizacija). VMWare Workstation omogućava rad s virtuelnim mašinama i na računarima bez hardverske podrške za virtuelizaciju. Instrukcije u virtuelnim mašinama se dijele na "osjetljive" i "neosjetljive". Neosjetljive se izvršavaju direktno na procesoru, dok se osjetljive dinamički

rekompajliraju, što može smanjiti performanse. Optimizacije, poput keširanja i JIT kompajliranja, poboljšavaju rad. Kao hipervizor tipa 2, koristi osnovni OS za komunikaciju s uređajima, što donosi široku kompatibilnost, ali bez garancija za drajvere.

VMWare Workstation ima tri komponente:

1. **VMX drajver** - Radi u Ring 0 i omogućava privilegovan pristup procesoru.
2. **VMM (Virtual Machine Monitor)** - Hipervizor koji radi iz kernel memorije, skriven od osnovnog OS-a.
3. **VMApp** - Korisnička aplikacija s GUI-jem za upravljanje virtuelnim okruženjem.

Ovo rješenje je popularno za razvojne svrhe, zbog balansa između performansi i fleksibilnosti.

---

34. Virtuelizacija- virtuelizovane komponente

Da bi se uspešno sproveo proces virtuelizacije, neophodne su sve osnovne komponente koje su prisutne u fizičkom računaru. Virtuelizacija memorije se koristi da upravlja pristupom memoriji od strane virtuelnih mašina, a predstavlja izazov jer operativni sistemi teže maksimalnom korišćenju memorije. Kada nema dovoljno radne memorije, može doći do korišćenja virtuelne memorije na sporijim uređajima, što dovodi do degradacije performansi, posebno kod hipervizora tipa 2, koji se bore za resurse sa korisničkim aplikacijama. Pristup I/O uređajima je privilegovan i zahteva dinamičko kompajliranje, što može dovesti do gubitka performansi. Korišćenjem paravirtuelizovanih drajvera može se poboljšati efikasnost, jer ovi drajveri direktno komuniciraju sa hipervizorom. Operativni sistem unutar virtuelne mašine prepoznaje sve standardne komponente, uključujući generičke uređaje, portove, mrežne kartice i hard diskove, a sadržaj virtuelnih diskova se obično čuva kao fajlovi unutar osnovnog operativnog sistema, što olakšava migraciju i bekap virtuelnih mašina. Ovi fajlovi dinamički zauzimaju prostor i rastu s vremenom, a emulacija CD i DVD uređaja se takođe sprovodi korišćenjem ISO fajlova.

ili ovaj tekst..

Jedan od uslova za praktičnu virtuelizaciju je postojanje svih osnovnih komponenata koje tipično postoje u stvarnom računaru i u virtuelizovanoj instanci. Ranije smo obradili problem virtuelizacije procesora kao osnovne komponente (uz podršku za hardverski potpomognutu virtuelizaciju ili ne), dok je problem virtuelizacije memorije prisutan nezavisno od hipervizora i većina hipervizora i koristi ugrađene sisteme za virtuelizaciju memorije da bi upravljala pristupom memoriji od strane virtuelnih mašina, na vrlo sličan način kao što to radi i kernel za standardne procese.

Kod virtuelizacije memorije postoji problem određivanja količine radne memorije koja je dostupna virtuelnim mašinama. U praksi operativni sistemi teže da što više koriste memoriju (jer već postoji i troši električnu energiju bez obzira da li je mi koristimo pa je šteta da je ne koristimo), najčešće na taj način što svu slobodnu memoriju koriste kao keš za sporu sekundarnu memoriju. Problemi nastaju u situacijama u kojima nema dovoljno primarne memorije i u kojima moramo da koristimo virtuelnu memoriju na sporoj sekundarnoj memoriji (swap fajl ili swap particije). Ovaj problem je naročito izražen u virtuelizovanim okruženjima jer sama virtuelna mašina možda nije svjesna da je zbog pritiska na memoriju dio njene radne memorije, za koju ona misli da se nalazi u RAM-u, zaista smješten u fajl na disku. U tom slučaju će kernel unutar virtuelne mašine pokušavati da prebacuje swap sa diska na disk u nadi da će se taj dio memorije naći i u stvarnoj radnoj memoriji. Sve navedeno degradira performanse do neupotrebljivosti. Poseban problem predstavljaju hipervizori tipa 2 jer se kod njih virtuelne mašine ne takmiče za resursa samo između sebe, nego i protiv korisničkih aplikacija koje su pokrenute u osnovnom operativnom sistemu.

Kada govorimo o pristupu I/O uređajima, on je uvijek osjetljiv (privilegovan) i svodi se na dinamičko kompajliranje (jer moramo efikasno da vršimo kontrolu nad pristupom stvarnim uređajima). Vrši se prebacivanje iz emuliranog svijeta u stvarni OS što stvara gubitke performansi, a dešava se jako često (jer često koristimo sekundarnu memoriju i periferije). U okviru tog procesa operacije niskog nivoa se pretvaraju u operacije višeg nivoa, na primjer, direktan pristup hard disku (prekidi i DMA) se prevodi u običan read() koji se opet realizuje i na stvarnom hard disku (prekidi i DMA). Da bi izbjegli naporno praćenje svih aktivnosti virtuelizovanog okruženja, moguće je

koristiti paravirtuelizovane drajvere čime poboljšavamo performanse. Ovi drajveri su "svjesni" da ne rade sa stvarnim periferijama i umjesto da pokušavaju koristiti prekide, uputiće hipervizoru "molbu" da im omogući pristup virtuelnom uređaju.
Sam operativni sistem unutar virtuelne mašine vidi sve standardne komponente koje su očekivane u klasičnom računaru: generički uređaji (tastatura, miš, itd), portovi (serijski, paralelni, USB), audio kartica (Sound Blaster kompatibilna ili neka slična), mrežna kartica (uz mogućnost upotrebe više mrežnih kartica u svakoj VM i pravljenje virtuelnim mrežnih infrastruktura), grafička kartica (VGA ili napredniji drajver), te hard disk (IDE/ATA, SCSI). Iako je moguće dozvoliti i direktan pristup disku ili particiji, obično se sadržaj virtuelnih diskova čuva u fajlovima u okviru fajl sistema osnovnog operativnog sistema. Na ovaj način je moguća jednostavna *off-line* migracija virtuelne mašine na drugi računar (iskopiramo fajlove na drugi računar), kao i bekap (iskopiramo fajlove na eksterni medij). Sami fajlovi obično dinamički zauzimaju neophodan prostor i čuvaju samo upisane informacije, te rastu vremenom (npr. virtuelni HDD od 1TB zauzima nekoliko KB prazan, ali i ~100GB ako čuva 100GB upisanih podataka). Na ovaj način radi i emulacija CD i DVD uređaja (koji obično koriste ISO fajlove kao medij).

---

35. Paravirtuelizacija

Paravirtuelizacija je pristup u virtuelizaciji koji modifikuje gostujući operativni sistem da bude "svestan" virtuelnog okruženja, što olakšava kontrolu hipervizora. Ovaj proces zahteva API, poznat kao hyper-API, koji omogućava komunikaciju između hipervizora i gostujućeg operativnog sistema putem hiper poziva. Umesto da se koriste prekidi za pristup periferijama, kernel koristi API pozive, što čini hipervizor jednostavnijim, sigurnijim i efikasnijim. Postoje dva glavna pristupa paravirtuelizaciji: prvi podrazumeva modifikaciju samo drajvera za teško virtuelizovane komponente, dok drugi zahteva izmene i samog kernela operativnog sistema. Drugi pristup omogućava veću efikasnost i sigurnost, jer gostujući kernel može da iskoristi hardverske resurse i implementira sigurnosne mehanizme koji se teško virtuelizuju. Ovaj koncept je široko usvojen među proizvođačima rešenja za virtuelizaciju kao što su VMware,

Microsoft, Xen i Google.

ili ovako:

Osnovni koncept paravirtuelizacije možemo iskazati na sljedeći način: Umjesto da konstantno nadziremo šta želi da uradi gostujući operativni sistem, mi ga modifikujemo na takav način da je "svjestan" da se izvršava u virtuelnoj mašini i da se ponaša na način koji nama olakšava kontrolu, a njemu izvršavanje.
Da bi ovaj pristup bio praktično upotrebljiv, potrebno je da hipervizor obezbijedi
i API koji može da koristi gostujući operativni sistem kako bi hipervizoru saopštio svoje potrebe.
Kako je namjena API-ja komunikacija sa hipervizorom, obično se naziva hyper-API, dok se pozivi , po ugledu na sistemske pozive, nazivaju hypercalls ili hiper pozivi.

Primjer upotrebe ovih poziva je: umjesto da kernel u virtuelnoj mašini poziva prekid da bi pristupio nekoj periferiji, on koristi API poziv i direktno nam predaje kontrolu da u njegovo ime uradimo potrebne zadatke.
Zašto to radimo?
Na ovaj način dobijamo mnogo jednostavniji
hipervizor jer više ne mora da sadrži dijelove i funkcionalnosti za načine upotrebe koji se teško virtuelizuju.

Samim tim, ovakvi hipervizori su sigurniji jer imaju manju kompleksnosti i manju izloženost napadačima.
S druge strane, performanse su bolje jer gost aktivno sarađuje sa nama umjesto da se bori za kontrolu nad periferijama.
Kako su prednosti očigledne, očekivano je i da gotovo svi proizvođači rješenja za virtuelizaciju većinu funkcionalnosti zasnivaju na ovom pristupu:
VMWare, Microsoft, Xen, Linux, Google, itd.

Kada govorimo o paravirtuelizaciji postoje dva osnovna pristupa:

· Izmjena samo drajvera za komponente koje je teško virtuelizovati - ovo je jednostavniji

pristup koji omogućava upotrebu gotovo neizmijenjenog operativnog sistema u virtuelnoj mašini jer sve što zaista radimo jeste instalacija dodatnih drajvera.
Ovaj pristup takođe i rješava većinu problema vezanih za performanse pri virtuelizaciji jer često najveći problem I jesu periferije i njihova efikasna emulacija.

· Izmjena i samog kernela operativnog sistema koji se izvršava u virtuelnoj mašini.

Sada se ne zadovoljavamo samo periferijama nego želimo i da osnovne funkcionalnosti kernela (raspoređivač, upravljanje memorijom, sigurnosni aspekti, itd) budu prilagođeni za izvršavanje u virtuelnoj mašini. Želimo da gostujući kernel bude u potpunosti "svjestan" da se ne izvršava na stvarnom hardveru nego u VM.

Na ovaj način omogućavamo još veću efikasnost i povećavamo sigurnost implementacije jer smo sada u stanju da u potpunosti iskoristimo i kompletnu hardversku podršku koja nam je možda na raspolaganju. Takođe, kako određene elemente sistema ne možemo efikasno virtuelizovati (sigurnosni mehanizmi ugrađeni u procesor ili matičnu ploču) na ovaj način možemo da obezbijedimo njihovu funkcionalnost i u virtuelizovanom okruženju koje mora biti prilagođeno za takav način rada.

---

36. KVM i QEMU

KVM (Kernel-based Virtual Machine) je modul jezgra koji omogućava Linuxu i FreeBSD-u da funkcionišu kao hipervizori, kombinujući karakteristike hipervizora tipa 1 i tipa 2. Ovaj sistem omogućava izvršavanje virtuelnih mašina (VM) bez gubitka funkcionalnosti operativnog sistema, podržavajući hardverski potpomognutu virtuelizaciju i razne arhitekture, uključujući x86-64 i ARM. KVM ne emulira kompletno okruženje; umesto toga, njegova funkcija je da pruži virtuelni CPU, memoriju i I/O interfejs, dok QEMU služi kao komplement koji završava proces emulacije.

KVM omogućava dinamičke promene broja procesora i količine memorije, kao i optimizaciju memorije deljenjem zajedničkih strana između virtuelnih mašina. Takođe podržava živu migraciju, što omogućava premestanje VM bez prekida rada. QEMU, kao "švajcarski nož" za emulaciju, podržava različite modove rada, uključujući emulaciju celog sistema i različite arhitekture.

Napredne funkcionalnosti QEMU uključuju zamrzavanje sistema, potpunu virtuelizaciju, JIT rekompajliranje (TCG), i emulaciju raznih periferija poput virtuelnih diskova, mrežnih kartica, i grafičkih uređaja. QEMU je fleksibilan i ne zahteva superuser privilegije za mnoge operacije, čime olakšava rad sa različitim arhitekturama.

Ili bez skraćivanja..:

KVM je skraćenica od Kernel-based Virtual Machine. Sam KVM ne predstavlja ni hipervizor tipa 1 ni tipa 2 jer zadovoljava kriterijume za oba. KVM je realizovan kao kernel modul koji omogućava kernelu da u potpunosti funkcioniše i kao hipervizor, bez gubitka drugih funkcionalnosti. To praktično znači da smo u situaciji u kojoj ispod hipervizora ne postoji drugi operativni sistem (znači tip 1), ali i u situaciji u kojoj je osim izvršavanja virtuelnih mašina moguće
isti sistem koristiti i za uobičajene namjene, a koristimo i već postojeće drajvere tog operativnog sistema (znači tip 2).

KVM podržava kernele Linux i FreeBSD operativnih sistema i zahtijeva postojanje hardverski podržane virtuelizacije i trenutno podržava x86-64, S/390, PowerPC, IA64 i ARM ISA.

Osim potpune virtuelizacije, za određene operativne sisteme postoji i podrška za paravirtuelizaciju u vidu
drajvera za mrežnu i grafičku karticu, disk kontroler, ali i druge drajvere. Kako je riječ o dijelu zvaničnog Linux kernela, KVM je izuzetno popularan i prisutan u velikom broju besplatnih i komercijalnih rješenja. Važno je napomenuti da KVM ne emulira kompletno okruženje i nije dovoljan za potpuni rad VM. Uloga KVM-a je da omogućava drugim projektima da završe kompletnu emulaciju (QEMU, Google crosvm), dok KVM obezbjeđuje virtuelni CPU, memoriju i IO interfejs (VirtIO).

Neke od mogućnosti koje nam KVM pruža su: dinamička promjena broja procesora (dodavanje ili oduzimanje broja logičkih jezgara koje su dostupna unutar virtuelne mašine bez zaustavljanja izvršavanja iste), dinamička promjena količine memorije (kroz balooning) i ušteda dupliranih memorijskih strana između virtuelnih mašina (tako da ako u 10 VM postoji isti kernel I skup biblioteka, KVM će u memoriji držati samo prvu sliku i reference na njih za preostale VM čime se smanjuje zauzeeć stvarne memorije i ubrazava rad), te podrška za živu migraciju koja nam omogućava da virtuelnu mašinu premjestimo sa jednog na drugi računar bez zaustavljanja izvršavanja iste.

Prirodni komplement KVM-a predstavlja QEMU - Švicarski nož funkcionalnosti emulacije. QEMU podržava različitr modove rada: user-mode emulacija, saradnja sa KVM-om, emulacija kompletnog sistema i proizvoljne arhitekture, itd. Za većinu funkcionalnosti nisu neophodne superuser privilegije što omogućava jednostavniji rad, naročito u slučaju upotrebe koji nam omogućava da za određeni izvršni fajl emuliramo okruženje potrebno za njegovo izvršavanje (npr.
posjedujemo izvršni fajl kompajliran za ARMv8.2 64-bitni procesor i želimo da ga izvršimo na x86-64 procesoru - moguće je uz QEMU). Sam QEMU ima podršku za brojne arhitekture: x86 i x86-64, ARMv7, ARMv8, MIPS, PowerPC, SPARC, RISC-V, itd.
Od naprednih funkcionalnosti treba istaći: zamrzavanje izvršavanja i čuvanje trenutne slike sistema (snapshot), podršku za potpunu virtuelizaciju (KVM i Intel HAXM) ili JIT rekompajliranje (koje se u tom slučaju naziva TCG), te vrlo naprednu emulacija raznorodnih periferija.
Kada govorimo o emulaciji periferije, QEMU podržava rad sa sljedećim: virtuelni diskovi u qcow2 formatu koji raste po potrebi, podržava copy-on-write i slojeve, kao i AES enkripciju; mrežne kartice uz podršku za NAT i proizvoljne mrežne topologije, te ugrađen fajl server (SMB); grafička kartica koja ne mora ni zaista postojati i omogućava udaljeni grafički pristup virtuelnoj mašini; kao i USB, Zvučna kartica, NVMe, serijski i paralelni port, itd.
---

37. Virtuelizacija na nivou operativnog sistema

Virtuelizacija na nivou operativnog sistema omogućava efikasno korišćenje resursa kroz paravirtuelizaciju, čime se rešavaju problemi klasične virtuelizacije. Ovaj pristup omogućava dinamičko upravljanje memorijom, efikasan nadzor izvršavanja aplikacija, i manji gubitak performansi, posebno pri emulaciji periferija.

Umesto korišćenja različitih operativnih sistema u virtuelnim mašinama (VM), virtuelizacija na nivou operativnog sistema se fokusira na korišćenje istog OS-a, čime se eliminira potreba za posrednikom (hipervizorom). Kernel hosta upravlja procesima iz VM kao zasebnim grupama, čime se zadržava ekvivalentnost i ne gubi performansa.

Prednosti ovog pristupa uključuju jednostavniju implementaciju, brže pokretanje, i efikasno upravljanje sekundarnom memorijom i periferijama. Međutim, mane uključuju ograničenje na korišćenje istog kernela, što može izazvati sigurnosne rizike.

Virtuelizacija na nivou operativnog sistema ima različite nazive, uključujući kontejneri, virtuelni privatni serveri (VPS), i jails, a danas se najčešće koristi termin "kontejner" kao ekstremni oblik ovog pristupa, pri čemu se obično koristi jedan kontejner za svaku aplikaciju, umesto više aplikacija unutar jedne VM.

---

38. Implementacija virtuelizacije na nivou operativnog Sistema

Implementacija virtuelizacije na nivou operativnog sistema potiče iz želje za izolacijom procesa unutar fajl sistema. Ovaj proces se prvobitno ostvaruje korišćenjem **chroot** mehanizma, koji menja root direktorijum za određeni proces, ali nije savršen sigurnosni alat. Zbog toga je razvijen **FreeBSD jail** 2000. godine, koji dodatno jača sigurnost i izolaciju procesâ.

Prva komercijalna implementacija virtuelizacije na nivou OS-a u Linuxu je **Virtuozzo** iz 2000. godine, a kasnije je 2005. godine objavljen **OpenVZ** kao open-source verzija. OpenVZ koristi modifikovani Linux kernel za izolaciju fajl sistema, korisnika, i resursa, uz mogućnost postavljanja kvota i kontrolu resursa.

Kako bi se integrisali principi OpenVZ u zvanični Linux kernel, razvijen je **Linux Containers (LXC)**. Iako se LXC retko koristi samostalno, često je osnova za druge alate kao što su **LXD** i **Docker**. LXC omogućava različite nivoe virtuelizacije, od jednostavnog chroot-a do složenijih rešenja.

Virtuelizacija se postiže deljenjem kernela u više particija i efikasnim nadzorom resursa. **Imenski prostori (Linux namespaces)** omogućavaju izolaciju različitih aspekata kao što su procesi, mrežni servisi i fajl sistemi, dok **kontrolne grupe (cgroups)** ograničavaju i prate upotrebu resursa od strane grupa procesa. Ovaj pristup se oslanja na UNIX filozofiju, bez dodavanja novih sistemskih poziva, a omogućava i **Checkpoint/Restore** funkcionalnost za snimanje i ponovno pokretanje stanja VM-a.

---

39. Docker

Docker je set alata za rad s kontejnerima koji olakšava upravljanje kontejnerima, iako ne uvodi nove tehnologije, već koristi postojeće (poput LXC, Linux namespaces i cgroups). Dostupan je kao open-source i komercijalno rešenje, sa različitim alatima od jednostavnih do kompleksnih sistema za klastere.

Funkcija Dockera se oslanja na **standardizovane slike (images)** kontejnera koje predstavljaju aplikacije ili servise. Svaka slika se sastoji od više slojeva; gornji slojevi sadrže izmene u odnosu na donje slojeve, omogućavajući efikasno vraćanje kontejnera u izvorno stanje jednostavnim restartovanjem. Ova slojevita organizacija smanjuje prostor potreban za skladištenje i ubrzava preuzimanje slika.

Pribavljanje slika funkcioniše slično kao kod repozitorijuma paketa na Linuxu, gde su kreirani repozitorijumi za čuvanje slika, slojeva i metapodataka. Pri ažuriranju, preuzimaju se samo slojevi s izmenama.

Docker ne koristi nove tehnologije, ali se prilagođava Windows sistemima korišćenjem **WSL2** ili **Hyper-V** virtuelnih mašina. Korisnički zahtevi se pretvaraju u poruke koje **docker daemon** obrađuje. Zbog složenosti rada, Docker se najčešće koristi na razvojnim stanicama, dok se na serverima često koristi okruženje poput **Kubernetes** za izvršavanje kontejnera. Za upravljanje zavisnim kontejnerima, Docker nudi alat **docker-compose**, koji olakšava rad s aplikacijama koje zahtevaju više kontejnera, kao što su izvršna okruženja, baze podataka i sistemi za nadzor.

40.**Razvoj tehnologija virtuelizacije**

Razvoj tehnologija virtuelizacije obuhvata progresiju od opštih pristupa sa slabijim performansama do specijalizovanih rešenja sa vrhunskim performansama, uključujući simulaciju, emulaciju, klasičnu virtuelizaciju, paravirtuelizaciju i virtuelizaciju na nivou operativnog sistema (kontejnere).

- **Simulatori** omogućavaju detaljan pristup simuliranim sistemima, ali obično rade sporije.

- **Emulatori** omogućavaju brži rad s procesorima u softverskom okruženju, ali bez detalja simulacije.

- **Virtuelizacija** omogućava korišćenje nemodifikovanih operativnih sistema u virtuelnim mašinama, s direktnim izvršavanjem instrukcija na procesoru.

- **Paravirtuelizacija** zahteva modifikovane sisteme za lakšu virtuelizaciju, ali smanjuje fleksibilnost.

- **Virtuelizacija na nivou operativnog sistema** koristi isti kernel na hostu i guestu, minimizirajući gubitke, ali sa strožim pravilima rada.

Ova progresija dovodi do gubitka fleksibilnosti, ali poboljšava performanse, omogućavajući upotrebu kontejnera za kompleksne tehnologije kao što su service mesh i sistemi za upravljanje obradom podataka.

41.**Multimedijalni operativni sistemi**

Multimedijalni operativni sistemi fokusiraju se na problem vremena odziva, koji mora biti unutar propisanih okvira kako bi se osiguralo korisničko iskustvo i funkcionalnost sistema. Ključni pojmovi uključuju:

- **Kašnjenje (latency)**: Vrijeme između zahteva i odgovora; višestruki faktori mogu doprineti kašnjenju.

- **Varijacija kašnjenja (jitter)**: Problemi u reprodukciji multimedijalnih sadržaja; rešenje uključuje korišćenje bafera, koji smanjuje varijacije, ali dodaje fiksno kašnjenje.

- **Reprodukcija multimedije**: Optimizacija skladištenja podataka i sinhronizacija audio i video zapisa (npr. interleaving u AVI formatu).

- **Digitalno-analogne konverzije**: Korišćenje D/A konvertora za reprodukciju zvuka uz očuvanje kontinuiteta.

- **Deadline**: Mogućnost postojanja soft i hard deadline-a; hard deadline znači gubitak funkcionalnosti u slučaju kašnjenja.

- **Real-time sistemi**: Deterministično ponašanje koje omogućava predvidljivo vreme operacija; primeri uključuju token ring pristup.

Kroz različite tehnike raspoređivanja i hijerarhijsko upravljanje zadacima, multimedijalni operativni sistemi osiguravaju optimalno korišćenje resursa i garantovano vreme odziva, što je ključno za aplikacije kao što su igre i interaktivni sadržaji.

---

42. Razvoj mobilnih platformi i operativnih sistema

Razvoj mobilnih platformi i operativnih sistema može se podeliti u tri glavne etape:

1. **Mobilni telefoni**: U ovoj fazi, telefoni su se koristili prvenstveno za glasovne pozive i SMS poruke. Interakcija je bila jednostavna, sa osnovnim menijima, malom radnom memorijom i niskim kvalitetom ekrana. Operativni sistemi su bili osnovni, bez podrške za dodatne aplikacije.

2. **Feature phone era**: Ovi uređaji su počeli podržavati multimedijalne sadržaje, poput audio i video zapisa, uz ograničenu podršku za internet. Email je postao popularan, a korisnici su mogli instalirati jednostavne aplikacije. Procesori su bili napredniji, a podržane su dodatne periferije poput Bluetooth slušalica.

3. **Smartphone era**: Ova faza je revolucionisala mobilne telefone, pretvarajući ih u multifunkcionalne uređaje. Pružaju pristup savremenim internet servisima, podržavaju rad sa multimedijom na napredan način (npr. editovanje 4K video zapisa), a aplikacije su postale ključni deo korisničkog iskustva. Operativni sistemi su postali neizbežni i igraju kritičnu ulogu u interakciji sa korisnicima i izboru aplikacija.

Ukratko, evolucija mobilnih platformi prati napredak tehnologije i promene u načinu korišćenja uređaja, s naglaskom na sveobuhvatne funkcionalnosti i korisničko iskustvo.

---

43. Savremeni mobilni operativni sistemi

Savremeni mobilni operativni sistemi karakterišu se dominacijom dva ključna sistema: Google Android i Apple iOS. Analizirajući period od 2007. do 2018. godine, primećuje se da je na početku ovog razdoblja najdominantniji operativni sistem bio Symbian, koji je koristio nezgrapan C++ razvojni okvir. Microsoft je pokušavao prilagoditi Windows operativni sistem malim ekranima, ali je izgubio trku za popularnošću. Blackberry je imao uspeh u poslovnom segmentu, ali nije uspeo da se prilagodi prelasku na ekrane osetljive na dodir.

S pojavom iPhone-a, došlo je do dramatičnih promena u mobilnom tržištu, gde su Android i iOS preuzeli kontrolu, dok su Symbian i Windows mobilni uređaji postali marginalizovani. Značajan aspekt ove promene je i uvođenje zvaničnih prodavnica aplikacija koje su olakšale korisnicima preuzimanje i ažuriranje aplikacija, a programerima omogućile efikasnu distribuciju i monetizaciju.

U pogledu profita, većina proizvođača ostvaruje malo ili nimalo profita, dok su Apple i Samsung glavni dobitnici. Huawei se suočio s izazovima zbog američkih sankcija, dok Xiaomi, iako je na drugom mestu po broju prodatih uređaja, i dalje beleži nizak profit.

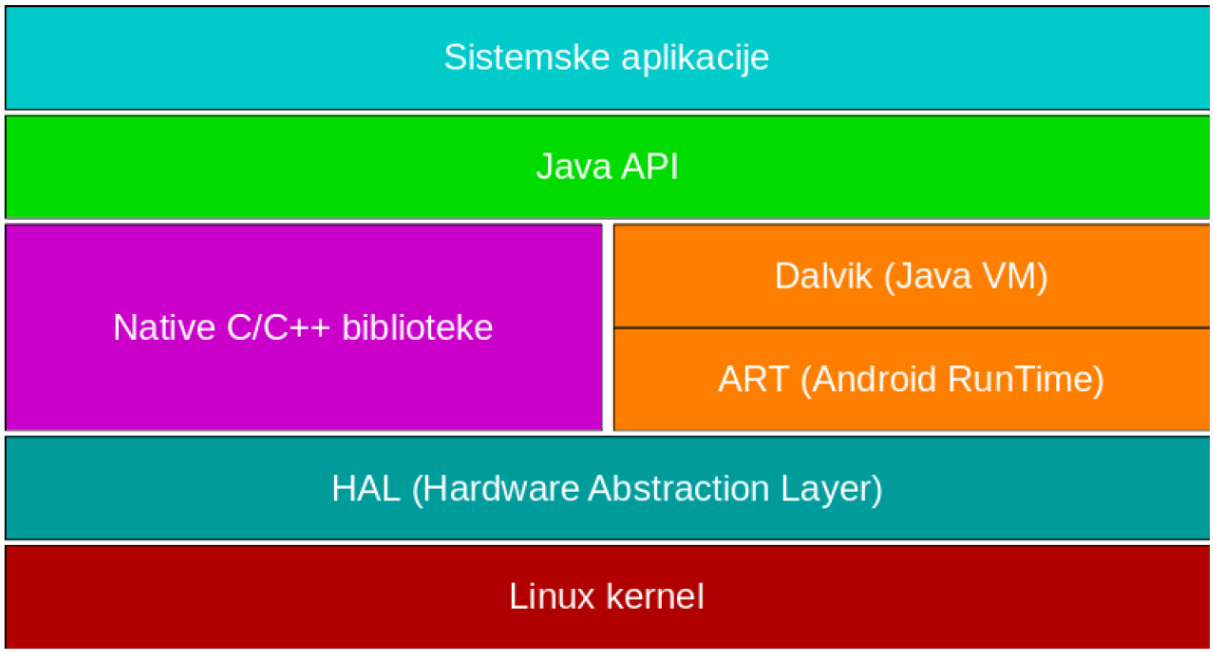
Pored Android-a i iOS-a, postoji i nekoliko drugih operativnih sistema poput Tizen-a (koji razvija Samsung) i WebOS-a (koji razvija LG), koji se koriste na pametnim uređajima i televizorima. Svi ovi sistemi koriste Linux kernel, što dovodi do sličnosti u njihovoj strukturi, ali stvara i zabrinutost zbog monokulture koja može ograničiti inovacije. Ovaj fenomen otvara mogućnosti velikim kompanijama da razviju sopstvena rešenja, poput Google-ovog Fuchsia operativnog sistema.

---

44. Android

Android je danas najpopularniji mobilni operativni sistem sa više od 80% tržišnog udela prema broju isporučenih uređaja. Ako se uzmu u obzir i drugi uređaji kao što su automobili, televizori i kućanski aparati, jasno je koliko je Android postao važan deo svakodnevnog života. Iako su zagovornici Linuxa dugo čekali "godinu Linuxa na desktopu", ona se verovatno nikada neće dogoditi, jer se Linux kernel već afirmisao na mobilnim uređajima.

Android se oslanja na prilagođeni Linux kernel, ali umesto GNU C biblioteke, koristi specijalno razvijenu ***bionic*** biblioteku. *Ova odluka omogućava proizvođačima da zadrže svoje izmene, što može biti korisno, ali takođe ograničava mogućnosti korisnika da menjaju funkcionalnosti uređaja nakon isteka podrške, koja obično traje dve do tri godine.*



Slika 7: Pojednostavljena arhitektura Adnroid operativnog sistema

Androidova arhitektura uključuje sloj za apstrakciju hardvera (HAL), koji omogućava standardizovan pristup različitim hardverskim komponentama, što je ključno za viši nivo, kao što je Java API. Zanimljivo je da Android koristi korisničke naloge na neobičan način, posmatrajući svaku aplikaciju kao poseban korisnički nalog. Ova odluka je olakšala razvoj, ali je takođe dovela do problema kada su se pojavile potrebe za više korisničkih naloga na jednom uređaju.

Aplikacije se izvršavaju na nekoliko načina. Native Development Kit (NDK) omogućava korišćenje C i C++ biblioteka, ali je najčešći pristup razvoj aplikacija korišćenjem Java programskog jezika u okviru Dalvik virtuelne mašine. Dalvik je vremenom napredovao, uključujući optimizacije kao što su Just-in-Time (JIT)

kompajliranje. Od verzije 5.0, Dalvik je zamenjen Android Runtime (ART), koji podržava i Ahead-of-Time (AOT) kompajliranje, čime se olakšava upravljanje memorijom i poboljšava performansa.

Google je u poslednjih nekoliko godina nastojao da ojača kontrolu nad Android ekosistemom, tako da se nove funkcionalnosti ne implementiraju direktno u operativni sistem, već u Google Play Services, koji nije otvorenog koda. Ovo je dovelo do situacije gde proizvođači, poput Samsunga, često nude aplikacije i prodavnice koje dolaze od različitih izvora, uključujući Google.

Na kraju, iako je Android uključivao dodatne slojeve emulacije, kao što je Java virtuelna mašina, ovo je omogućilo programerima da kreiraju aplikacije koje su funkcionalne na različitim arhitekturama procesora. U trenutku kada je Android razvijen, nije bilo jasno koja će arhitektura dominirati, što je dovelo do odabira pristupa sa međuslojem virtuelne mašine. Danas je jasno da su neki od izabranih pristupa možda bili suboptimalni, posebno s obzirom na to da su moderni jezici poput Kotlin-a i razvojna okruženja kao što su React Native i Flutter postali sve popularniji.

---

45. IOS

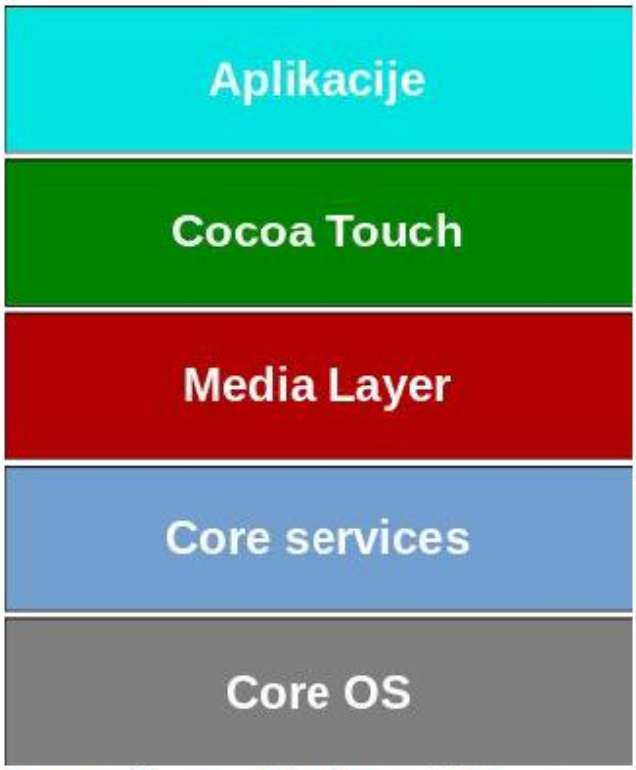
iOS je specifičan mobilni operativni sistem jer Apple kontroliše i hardver i softver, omogućavajući potpunu optimizaciju i sinhronizaciju između njih.

Pošto Apple proizvodi oba, nema rizika od nepredviđenih promena u specifikacijama, što vodi ka boljim performansama i kontrolisanoj integraciji.

iOS koristi mikrokernelski pristup, za razliku od monolitnog kernela kao što je Linux, i zbog toga je efikasniji. Sistem je organizovan u slojevima – osnovne servise, medijski sloj i sloj za interakciju s ekranom na dodir, dok se aplikacije nalaze na vrhu.

Instalacija aplikacija moguća je isključivo putem App Store-a, čime Apple zadržava kontrolu i ostvaruje značajan profit.

Programeri nemaju alternativne opcije prodavnica, što donosi problem potencijalne cenzure i Appleove provizije od 30% na transakcije.



Slika 8: Pojednostavljena arhitektura iOS operativnog sistema