



CS120 - ORGANIZACIJA RAČUNARA

Sloj operativnog sistema

Lekcija 10

PRIRUČNIK ZA STUDENTE

CS120 - ORGANIZACIJA RAČUNARA

Lekcija 10

SLOJ OPERATIVNOG SISTEMA

- ✓ Sloj operativnog sistema
- ✓ Poglavlje 1: Virtuelna memorija
- ✓ Poglavlje 2: Straničenje
- ✓ Poglavlje 3: Segmentacija
- ✓ Poglavlje 4: Virtuelna memorija Core i7
- ✓ Poglavlje 5: Hardverska virtualizacija
- ✓ Poglavlje 6: Pokazne vežbe
- ✓ Poglavlje 7: Zadaci za samostalni rad
- ✓ Poglavlje 8: Domaći zadatak
- ✓ Zaključak

Copyright © 2017 – UNIVERZITET METROPOLITAN, Beograd. Sva prava zadržana. Bez prethodne pismene dozvole od strane Univerziteta METROPOLITAN zabranjena je reprodukcija, transfer, distribucija ili memorisanje nekog dela ili čitavih sadržaja ovog dokumenta., kopiranjem, snimanjem, elektronskim putem, skeniranjem ili na bilo koji drugi način.

Copyright © 2017 BELGRADE METROPOLITAN UNIVERSITY. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, without the prior written permission of Belgrade Metropolitan University.

✓ Uvod

UVOD

Sloj Operativnog sistema (engl. operating system) je pozicioniran između sloja arhitekture skupa instrukcija ISA i sloja asemblera.

U ovoj lekciji ćemo detaljno govoriti o sloju Operativnog sistema

Ovaj sloj pozicioniran je između sloja arhitekture skupa instrukcija i sloja asemblera.

Sloj Operativnog sistema (engl. *operating system*) sa stanovišta programera na nivou **ISA** i izvan njega dodaje brojne nove instrukcije i mogućnosti.

Operativni sistem se po pravilu implementira softverski mada teoretski nema razloga da se implementira i hardverski.

Nivo na kome se implementira operativni sistem naziva se mašina operativnog sistema **OSM** (engl. *Operating System machine*).

U samoj lekciji obradićemo teme virtuelne memorije, tehnike koju primenjuju mnogi operativni sistemi kao i različite metode za upravljanje memorijom.

Te šeme mogu biti od krajnje jednostavnih do veoma složenih kao što je **straničenje (paging)** i **segmentacija**.

U lekciji su obrađeni slučajevi **virtuelne memorije** kao i **hardverska virtuelizacija** na Core i7 procesorima.

✓ Poglavlje 1

Virtuelna memorija

OPERATIVNI NIVO

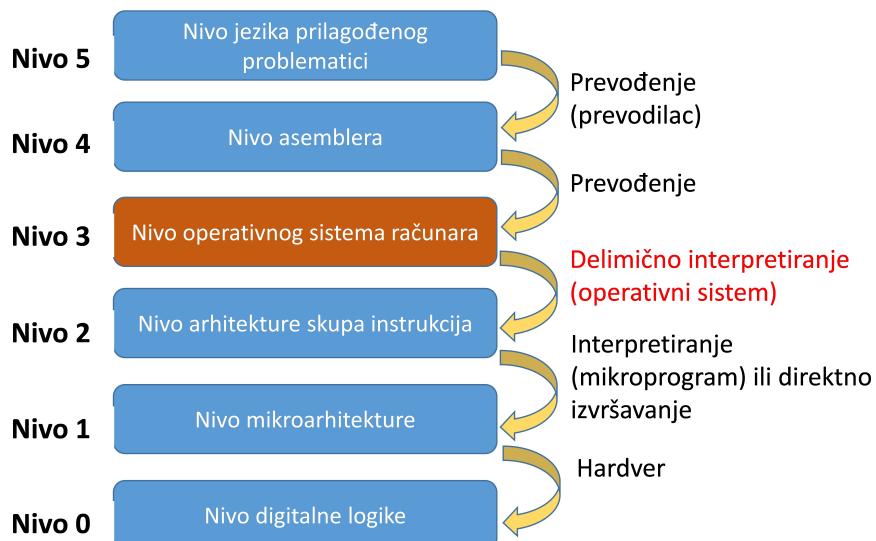
Nivo Operativnog sistema je iznad sloja arhitekture skupa instrukcija ISA

Nivo Operativnog sistema se uvek interpretira.

Kada korisnički program izvrši neku instrukciju **Nivoa Operativnog sistema OSM** (na primer pročita podatke iz neke datoteke), operativni sistem će raditi korak po korak, praktično kao što bi i mikroprogram izvodio instrukciju **ADD** korak po korak.

Kod slučaja kada program izvršava neku **instrukciju nivoa ISA**, nju direktno izvodi prethodni nivo mikroarhitekture bez ikakve pomoći operativnog sistema.

Operativni sistem se nalazi između nivoa Arhitekture skupa instrukcija i nivoa Operativnog sistema



Slika 1.1 Računar sa šest nivoa i načinom izvršavanja [Izvor:Autor]

ORGANIZACIJA MEMORIJE,

U toku razvoja računarskih sistema iznos glavne memorije instalirane u računaru se povećavao,

U toku razvoja računarskih sistema iznos glavne memorije instalirane u računaru se povećavao, ali se takođe i obim programa u poređenju sa dostupnom instaliranom memorijom sve više povećavao. Prvi pristup da se premosti ograničenje obima memorije zasniva se na primeni tehnike preklapanja (**overlays**).

Smisao se sastoji u deljenju programa na nekoliko delova. Jedan od ovih delova je uvek prisutan u memoriji i upravlja punjenjem drugih delova sekundarne memorije (disk, traka, itd.) u glavnu memoriju.

Ostali delovi se mogu puniti u glavnoj memoriji, pri čemu oni koriste istu memorijsku oblast, kao i prethodno korišćeni delovi, pa zbog toga kažemo da dolazi do preklapanja.

Organizacija memorije je ključno pitanje za memoriju sa slučajnim pristupom. Organizacija: fizički raspored bitova da bi se formirale reči.

Ključ uspeha memorijske hijerarhije je da podaci i instrukcije mogu da se rasporede tako da u najvećem delu vremena budu raspoloživi, kada su potrebni, na gornjim nivoima hijerahije.

Kada sa računarskim sistemom istovremeno radi veći broj korisnika smanjuje se deo glavne memorije koji je dodeljen svakom korisniku, a sa druge strane neophodno je uvesti neke mehanizme zaštite, koji će štititi (zaokruživati kao celinu) aktivnosti jednog korisnika od drugog.

Na koncept virtuelne memorije ukazaćemo sa dva aspekta:

Prvi se odnosi na translaciju adresa (određuje kako se adrese generisane od strane procesa, koji se izvršava, prevode u memorijske adrese) **a drugi na radni model** (na kojim se politikama zasniva **upravljanje memorijom** - MM).

REALIZACIJA VIRTUELNE MEMORIJE

Virtuelna memorija je tehnika koja dozvoljava izvršavanje procesa koji ne moraju biti kompletno u memoriji.

Jedna od glavnih prednosti ove šeme da programi mogu biti veći od veličine fizičke memorije .

Virtuelna memorija pravi apstrakciju od glavne memorije u jednu ogromnu logičku memoriju i oslobađa programera od bilo kakvih limita u pogledu veličine memorije.Takođe omogućava korisniku da lako deli datoteke i adresni prostor, a obezbeđuje se i efikasan mehanizam za kreiranje procesa.

Tehnika virtuelne memorije, koja omogućava da se izvrši program koji je samo delimično u memoriji, donosi sledeće dobre osobine:

1. Program nije više ograničen količinom raspoložive fizičke memorije. Programeri mogu da pišu programe, podrazumevajući ekstremno veliki adresni prostor, doduše virtuelni, ali će svefunkcionisati bez obzira na veličinu programa.

2.Zbog toga što svaki proces može zauzimati manje fizičke memorije, osetno više procesa mogu da se nađu u memoriji, povećavajući tako CPU iskorišćenje i propusni moć

(*through put*), a bez povećanja vremena odziva (*response time*) i vremena izvršavanja (*turnaround time*).

3. Očekuje se manje I/O operacija za punjenje ili razmenu korisničkih programa u memoriju, tako da bi programi trebalo da rade brže.

Virtuelna memorija predstavlja razdvajanje korisničke logičke memorije od fizičke memorije. Ta separacija omogućava da ekstremno veliki programi mogu da rade čak i na vrlo malim fizičkim memorijama. Sa virtuelnom memorijom, programeru je drastično olakšan posao, ne vodi računa o veličini memorije, ne razbija program na overlay komponente, pa može sa se koncentriše samo na programiranje.

Virtuelna memorija omogućava da se datoteke i memorija dele između različitih procesa na isti način kao kod deljenja stranica (*pages*) u straničnoj tehnici (*paging*), što omogućava uštedu memorije i povećanje performansi.

Virtuelna memorija se najčešće realizuje po tehnici učitavanja stranica prema potrebi DP (*demand paging*), a može se upotrebljavati i segmentacija, kao i njihove kombinacije.

Postoji tehnika koja se zove učitavanje segmenata prema potrebi (*demand segmentation*), ali su njihovi algoritmi za učitavanje i zamenu segmenata u memoriji mnogo kompleksniji u odnosu na stranice, pa ovu metodu nećemo obrađivati.

VIRTUELNA MEMORIJA OSNOVE

Virtuelna memorija je tehnika upravljanja memorijom koja se implementira koristeći i hardver i softver.

Memorije su na početku razvoja bele male i jako skupe.

U to vreme programeri su najveći deo vremena provodili pokušavajući da smeste programe u veoma oskudne po kapacitetu memorije.

Ovaj problem se obično rešavao na taj način što su programeri svoje programe delili na veći broj **segmenata** (engl. *overlays*), veličine koja može da stane u memoriju.

Pri pokretanju programa prvo se učitavao a nakon toga i izvršavao prvi segment. Po njegovom završetku očitavao se i izvršavao drugi segment i tako redom.

Programeri su bili odgovorni za razbijanje programa na segmente, za poziciju pojedinih segmenata u sekundarnoj memoriji, za prenos segmenata između sekundarne i glavne memorije i za upravljanje čitavim postupkom bez pomoći računara, što je zahtevalo jako puno posla.

Godine 1961. grupa naučnika sa **Univerziteta u Mančesteru (Engleska)**, predložila je metodu automatskog smenjivanja segmenata programa, za čiji rad čak programeri nisu ni morali da znaju.

Ova metoda nazvana je virtuelnom memorijom.

Virtuelan memorija (engl. virtual memory) je tehnika koja dozvoljava izvršavanje procesa čiji delovi mogu biti smešteni na sekundarnim memorijama, tj diskovima.

Virtuelna memorija formira apstrakciju u vidu logičke memorije, koju čine radna memorija i sekundarna memorija i razdvaja korisničku logičku memoriju od fizičke.

Količina raspoložive fizičke memorije više ne ograničava program, pa programeri mogu da pišu programe bez korišćenja tehnike **preklapanja** (**overlay**).

Koncept virtuelne memorije omogućava smeštanje osetno većeg broja procesa u memoriju (konkretno delova procesa), čime se povećavaju iskorišćenje i propusna moć procesora, a bez povećanja vremena

odziva (**response time**) i vremena izvršavanja (**turnaround time**).

Virtuelna memorija omogućava deljenje datoteka i memorije između različitih procesa na isti način kao i kod deljenja stranica, što omogućava uštedu memorije i poboljšanje performansi.

VIRTUELNA MEMORIJA I ADRESIRANJE

Kod virtualne memorije program je podeljen na delove fiksne dužine kojima automatski upravljaju hardver i operativni sistem.

Mehanizam se zove virtualna memorija jer se programima na raspolaganje daje prividna memorija koja zapravo ne postoji kao fizička memorija.

Kao što je već rečeno, fizička memorija je jednodimenzionalni niz koji se sastoji od **uzastopnih (susednih) stanica**. Fizička adresa jednoznačno određuje mesto gde se određena stanica nalazi u fizičkoj memoriji. Skup ovih adresa koje se direktno odnose na fizičku memoriju, naziva se fizički adresni prostor. S druge strane, svaki program ima svoj virtualni adresni prostor koji se sastoji od niza međusobno poređanih virtualnih adresa. Ove adrese koriste se kao reference na stvarne lokacije u fizičkoj memoriji. Nazivaju se virtualnim jer je moguće, prilikom izvođenja programa, da traženi sadržaj kojeg virtualne adrese referenciraju još nije učitan u fizičku memoriju. Na ovaj način program dobija iluziju da ima na raspolaganju „**neograničenu**“ količinu fizičke memorije.

Stanice u fizičkoj memoriji koje virtualne adrese referenciraju mogu se nalaziti na različitim lokacijama i ne moraju biti uzastopno poređane.

Dakle, podela na virtualni i fizički adresni prostor omogućila je pokretanje programa koji su veći od dostupne fizičke memorije i razvoj virtualne memorije kao mehanizma upravljanja memorijom.

Kako fizička memorija ne razume virtualne adrese one moraju biti prevedene u fizičke adrese. Za vreme izvođenja programa, odnosno kada procesor obrađuje program, posebni deo hardvera prevodi virtualne adrese u fizičke adrese.

Kao deo prevođenja, hardver takođe proverava je li sadržaj na traženoj adresi učitan u fizičku memoriju. U slučaju da je učitan, pristup će biti izведен. Ako sadržaj na adresi nije učitan

u memoriju, hardver će izdati prekid (engl. **interrupt**) kojeg će prihvatiti i obraditi operativni sistem.

Za vreme učitavanja program je zaustavljen i u ovom slučaju drugi programi mogu koristiti procesor. Samo nakon što je sadržaj na adresi učitan, prekinuti program može nastaviti sa izvršavanjem.

Dve su osnovne metode implementacije virtualne memorije.

To su straničenje i segmentacija

▼ Poglavlje 2

Straničenje

METODA STRANIČENJA

Straničenje čini jednu od osnovnih metoda implementacije virtualne memorije

Straničenje (engl. **paging**) je metoda implementacije virtualne memorije gde računar upisuje i čita podatke u sekundarnoj memoriji za korišćenje u **glavnoj** (radnoj) memoriji. Drugim rečima, ovo je metoda automatizacije preklopnog načina upotrebe radne memorije.

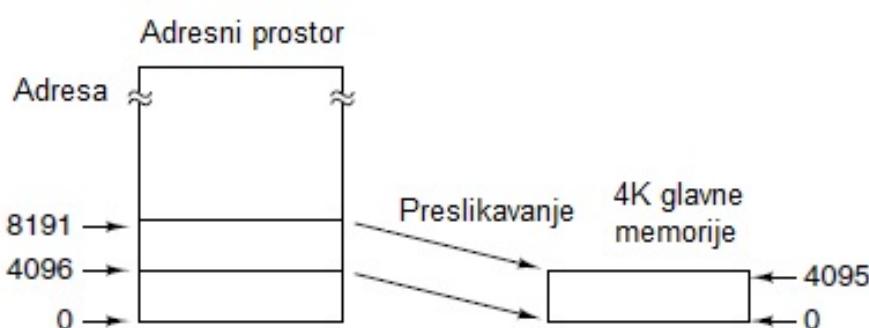
Osnovna svrha ovog mehanizma je da omogući pokretanje programa koji su veći od dostupne fizičke memorije, ali s bitnom razlikom da je celi posao realizacije ovog mehanizma prepusten operativnom sistemu koji radi sa delovima programa fiksne veličine.

Straničenje deli virtualni adresni prostor na blokove, koji se nazivaju stranice, pri čemu je svaka stranica istog obima.

Glavna memorija, kojoj se pristupa preko realnog adresnog prostora, deli se na blokove istog obima koji se zovu okviri stranica.

Translacija virtualne adrese u realnu adresu obavlja se pomoću tabele transliranja koja se zove tabela straničenja. Svaki proces ima sopstvenu tabelu straničenja, lociranu bilo gde u memoriji koja se adresira preko регистра tabele straničenja.

Za ovakvo preslikavanje adresa i adresnog prostora u stvarne memorijske lokacije, računar s memorijom od 4 KB koji nema virtualnu memoriju koristi fiksno preslikavanje adresa između 0 i 4095 u 4096 stvarnih memorijskih reči.



Slika 2.1 Preslikavanje pomoću koga virtualne adrese se prevode u adrese glavne memorije [Izvor:
Autor]

Ako program koristi adresu između 8192 i 12287, kod računara sa virtuelnom memorijom sam postupak preslikavanja je sledeći:

1. Sadržaj glavne memorije biće snimljen na disk.
2. Na disku će biti pronađene reči u intervalu između 8192 i 12287.
3. Reči iz intervala od 8192 do 12287 učitaće se u glavnu memoriju.
4. Mapa adresa biće tako izmenjena da preslikava adrese od 8192 do 12287 u memorijske lokacije od 0 do 4095.
5. Izvrsavanje će se nastaviti kao da je sve uobičajeno.

IMPLEMENTIRANJE STRANIČENJA

Kod straničenja, celi skup virtualnih adresa, podeljen je na delove nazvane stranice (engl. page).

Svaka stranica je jednake veličine i svaka logička adresa nalazi se na tačno jednoj stranici.

Analogno, fizička memorija podeljena je na **okvire** (engl. **frames**). Skup svih okvira nekog procesa zovemo njegovim **radnim skupom** (engl. **working set**).

U jedan okvir fizičke radne memorije može se smestiti jedna stranica logičkog adresnog prostora. Svaki okvir veličine je kao jedna stranica.

Veličinu stranice zapravo definiše veličina okvira, a ona obično iznosi od 1 KB do 8 KB, a najčešće je 4 KB za 32 bitni računarski sistem.

Slika 2. pokazuje jednu takvu konfiguraciju, koji ima logički adresni prostor veličine 64 KB. U ovom slučaju nalazi se 32 KB fizičke memorije i veličina stranice je 4 KB. Može se videti kako je logički adresni prostor podeljen na 16 stranica, a fizička memorija na 8 okvira.

Page	Virtual addresses	Page frame	Physical addresses
15	61440 – 65535	7	28672 – 32767
14	57344 – 61439	6	24576 – 28671
13	53248 – 57343	5	20480 – 24575
12	49152 – 53247	4	16384 – 20479
11	45056 – 49151	3	12288 – 16383
10	40960 – 45055	2	8192 – 12287
9	36864 – 40959	1	4096 – 8191
8	32768 – 36863	0	0 – 4095
7	28672 – 32767		
6	24576 – 28671		
5	20480 – 24575		
4	16384 – 20479		
3	12288 – 16383		
2	8192 – 12287		
1	4096 – 8191		
0	0 – 4095		

(a)

(b)

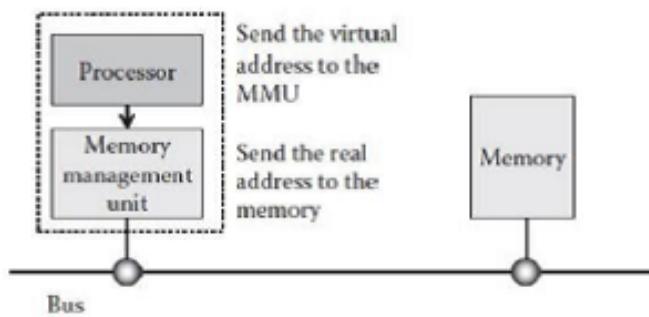
Slika 2.2 a) Prvih 64KB prostora virtuelnih adresa podeljenih u 16 stranica od po 4KB b) 32KB glavne memorije podeljena u osam okvira za stranice od po 4KB [Izvor: Autor]

JEDINICA ZA UPRAVLJANJE MEMORIJOM

Svaki računar s virtuelnom memorijom ima uređaj za preslikavanje iz virtuelnih adresa u fizičke.

Kako memorija prepoznaje samo fizičke adrese, a ne virtualne, one moraju biti predate prilikom pristupa podacima. Svaki računar sa virtualnom memorijom ima uređaj koji prevodi virtualne memorijske adrese u fizičke memorijske adrese. Ovaj uređaj zove se jedinica za upravljanje memorijom (engl. **MemoryManagement Unit - MMU**).

Obično je dio procesora. MMU je odgovoran za upravljanje komunikacijom između procesora i memorije kao i za prevođenje virtualnih adresa koje su unete u instrukcijama u stvarne adrese u memoriji (Slika 3.).

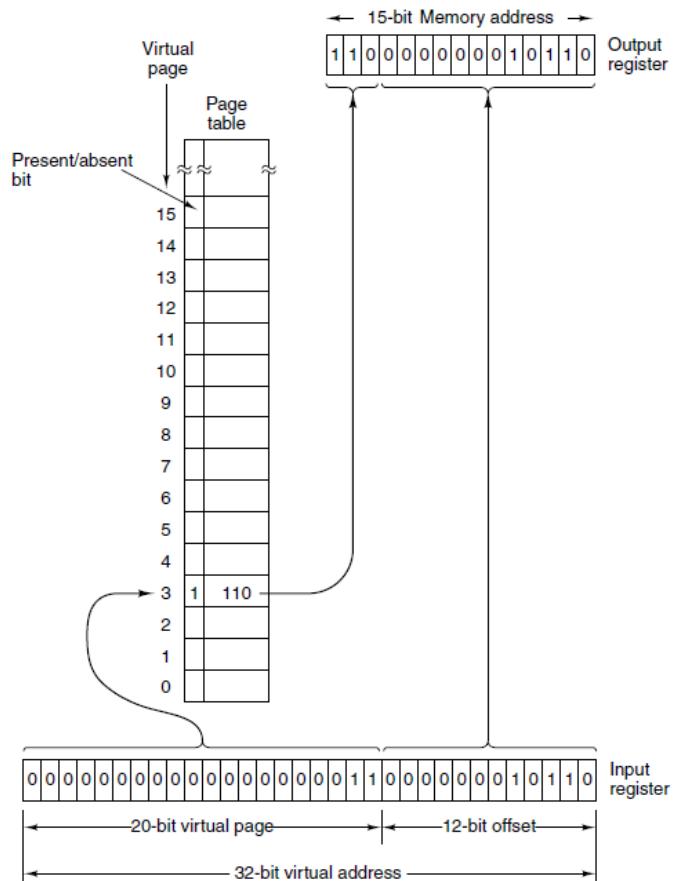


Slika 2.3 Prevođenje virtuelnih adresa u fizičke adrese [Izvor: Autor]

Na Slici 4. MMU prevodi 32-bitne virtualne adrese u 15-bitne fizičke adrese stoga treba 32-bitni ulazni registar i 15-bitni izlazni registar.

Kada MMU primi 32-bitnu virtualnu adresu, on ju podeli u 20-bitni broj stranice i 12-bitno odstojanje unutar stranice (zato što su stranice u primeru 4 KB).

Broj stranice se koristi kao indeks, odnosno kao referenca za pristup stranici unutar tabele stranice. Na slici broj stranice je 3 stoga je taj broj izabran za ulaz u tabelu stranica.



Slika 2.4 Formiranje adrese u glavnoj memoriji od virtuelne adrese [Izvor: Autor]

STRANIČENJE NA ZAHTEV

Slično kao kod straničenja, straničenje na zahtev deli program na stranice. Prilikom pokretanja programa u radnu memoriju se smeštaju samo one stranice koje su neophodne u prvom trenutku

Najpopularniji način implementacije virtuelne memorije, koji se u praksi pokazao kao najefikasniji, je **straničenje na zahtev** (Demand Paging).

Slično kao kod straničenja, ovaj pristup deli program na stranice.

Prilikom pokretanja programa u radnu memoriju se smeštaju samo one stranice koje su neophodne u prvom trenutku.

Kada se neka stranica zatraži pri izvršavanju procesa, prvo se proverava da li je ta stranica već u memoriji, pa ako nije pronalazi se na sekundarnoj memoriji i prebacuje u radnu.

Međutim ako tražena stranica nije u memoriji i mora biti učitana iz diska, MMU izdaje „**PageFault**“ **prekid** (engl. **interrupt**) kojeg operativni sistem registruje, obrađuje i prebacuje stranicu u slobodni okvir.

Istovremeno upisuje njenu novu fizičku adresu u tabelu stranice.

Takođe, ako je potrebno pritom se uklanja neka od drugih stranica koja je već učitana u okvir i najmanje se koristi kako bi se otvorio prostor (okvir) za novu stranicu.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

ZAMENA STRANICA

Zamena stranice je veoma bitna za straničenje na zahtev jer ona omogućava njenu realizaciju i efikasnost.

Čak i u slučaju optimalnog izbora stranica za procese, radna memorija može postati značajno popunjena nakon određenog vremena rada sistema.

Problem nastaje kada je potrebno učitati stranicu potrebnu za izvršavanje nekog od procesa, a pri tome ne postoji slobodan memorijski okvir.

Jedno od rešenja ovog problema podrazumevaju da se prekine onaj proces koji traži učitavanje stranice i kasnije ponovo pokrene od početka.

Drugi način je da se sve stranice procesa prebace na spoljašnju memoriju i oslobode svi okviri koje je proces zauzimao.

Rešenje koje se najčešće primenjuje je zamena stranice.

Kada je memorija popunjena, a potrebno je učitati novu stranicu, ideja je da se po nekom kriterijumu pronađe okvir ("žrtva") koji će se isprazniti kako

bi se u njega smestila nova stranica. Izabrana stranica se iz okvira prebacuje na sekundarnu memoriju i pri tome se ažurira tabela stranica tako da odgovara trenutnom stanju u sistemu.

Na ovaj način se oslobođa okvir za učitavanje nove stranice.

Postupak zamene stranice na kojima se bazira praktično ostvarenje postupka odabira stranica koje će biti izbačene iz svog okvira kada to bude potrebno, se može predstaviti na osnovu sledećih algoritama:

- **LRU strategija**, koja podrazumeva izbacivanje stranice koja se u prošlosti najmanje koristila (**LRU** - engl. **least recently used**).
- **FIFO strategija**, koja podrazumeva izbacivanje stranice koja je najviše vremena provela u radnoj memoriji (**FIFO** engl. **first-in,first-out**).
- **Optimalna strategija (OPT)**, koja podrazumeva izbacivanje stranice za koju se smatra da se ubuduće neće koristiti.

Zamena stranice je veoma bitna za straničenje na zahtev jer ona omogućava njenu realizaciju i efikasnost.

Uz pomoć dobro realizovane zamene stranica postiže se razdvajanje logičke memorije od fizičke memorije

PRIMER #1 STRANIČENJA

Zamislimo da imamo memoriju od 32 bajta, definišimo stranice veličine 4 bajta, što znači 8 stranica fizičke memorije.

Svaka logička adresa, koju generiše CPU se deli na 2 dela:

Broj stranice p (Page number p) - koristi se kao indeks i tabeli stranica koja sadrži baznu adresu fizičke stranice, okvira. Bazna adresa predstavlja viši deo adrese.

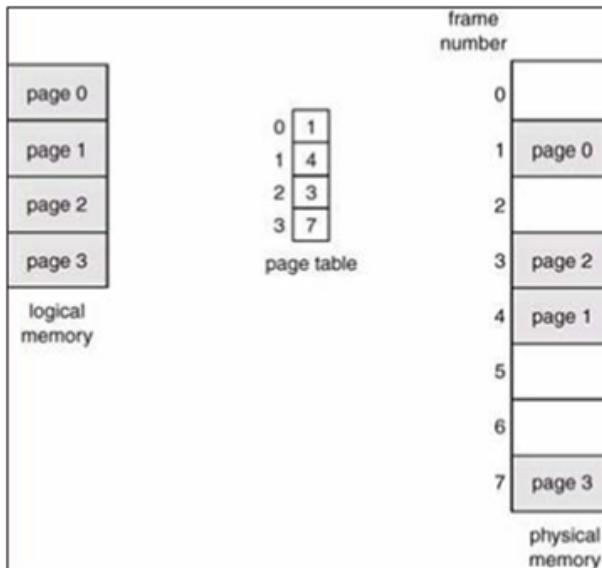
Pomeraj unutar stranice (Page offset d): definiše položaj u odnosu na samu stranicu i u kombinaciji sa baznom adresom (čisto sabiranje) definiše punu fizičku adresu koja se šalje memorijskoj jedinici. Naglasimo da je osnovna adresibilna jedinica bajt i da je pomeraj isti i za logičku i fizičku adresu. Ako je veličina logičkog adresnog prostora 2^m , a veličina stranice 2^n , tada viši deo adrese dužine m - definiše broj stranice, odnosno bazni viši deo adrese, a najnižih n bita adrese predstavljaju pomeraj unutar stranice.

Tehniku straničenja demonstriraćemo na jednom minijaturnom primeru. Zamislimo da imamo memoriju od **32 bajta**, definišimo stranice veličine **4 bajta**, što znači 8 stranica fizičke memorije. U ovom slučaju $m=6$, a $n=2$. Uzmimo korisnički proces koji zauzima 4 logičke stranice, sa logičkim adresama: stranica 0 (0-3), stranica 1 (4-7), stranica 2 (8-11) i stranica 3 (12-15).

Interpretirajmo neke logičke adrese.

Logička adresa 0, ima broj logičke stranice 0 i pomeraj 0. Pomoću broja logičke stranice ulazimo u tabelu stranica i nalazimo da je ona smeštena u okvir broj 1. I tako redom logička stranica 1 je smeštena u okvir 4, logička stranica 2 u okvir 3 i logička stranica 3 u okvir 7.

Na primer logička adresa 5 se mapira na sledeći način: adresa 5 je logička stranica 1 sa pomerajem 1, a na osnovu tabele stranica to okvir 4 sa pomerajem 1, adresa je $16+1=17$.



Slika 2.5 Primer #1 straničenja [Izvor: Autor]

PRIMER #2 STRANIČENJA

Svako straničenje predstavlja dinamičku relokaciju, a tabela stranica predstavlja relokacioni registar za svaki okvir fizičke memorije.

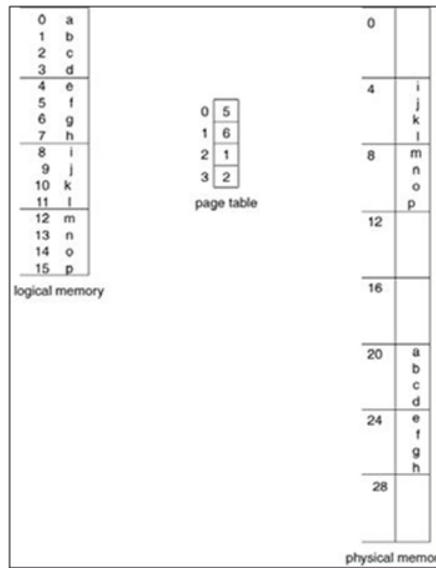
Imamo 16 bajtova u kojima je poređano prvih 16 slova abecede .

U logičkom kontinualni, a u fizikom razbacani. Obratite pažnju, ovde su adrese na bajt nivou. Svako straničenje predstavlja dinamičku relokaciju, a tabela stranica predstavlja relokacioni registar za svaki okvir fizičke memorije. Na ovaj način, straničenje potpuno eliminiše eksternu fragmentaciju, svaki okvir se može iskoristiti, kao deo bilo kog procesa. Međutim može se pojaviti, izvesni stepen interne fragmentacije, zato što deo stranice može ostati neiskorišćen i to se uvek dešava kada proces zahteva količinu memorije, koja je različita od celog broja stranica a procesu mora da se dodeli ceo broj stranica.

Minimalan gubitak je 0, a najveći gubitak imamo kada se zahteva količina memorije ($N * \text{page} + 1$) kada zbog jednog bajta gubimo celu stranicu-1, tako da je uprošćeno rečeno, srednji gubitak memorije po procesu iznosi $1/2$ stranice. To ukazuje da stranice treba da budu što manje, ali sa malim stranicama pojavljuje se drugi problem, a to je pretraživanje. Za svaku stranicu imamo jedan ulaz u tabeli stranica, pa sa malim stranicama njihov ukupan broj postaje veliki, pa imamo veliku tabelu stranica, koja unosi ozbiljne probleme u realizaciji i pretraživanju, preciznije, povećava se kašnjenje prilikom mapiranja. U slučaju korišćenja swap tehnike sa stranicama, poželjnije su veće stranice zato što su disk I/O zahtevi efikasniji kada su transferi veći, a najgora situacija za disk je kada radi sa malim blokovima podataka.

Generalno gledano, procesi, strukture podataka i veličine memorije su stalno rasle u vremenu, pa su rasle i veličine stranica, a radi se na realizaciji sistema sa promenljivim veličinama stranica.

Tabele stranica su 32 bitne uglavnom, 4 bajta po jednom ulazu, mada teže ka 64 bitnim strukturama. Na primer ako je tabela stranica 32-bitna i veličina stranica 4KB, može se adresirati $2^{32} \times 4 \times 2^{10} = 16\text{TB}$ fizičkog adresnog prostora.



Slika 2.6 Primer #2 straničenja [Izvor: Autor]

✓ Poglavlje 3

Segmentacija

METODA SEGMENTACIJE

Segmentacija (engl. segmentation) je metoda implementacije virtualne memorije.

Segmentacija (engl. **segmentation**) je metoda implementacije virtualne memorije.

Može biti korištena sama za sebe, ili u kombinaciji sa straničenjem.

U svojoj osnovnoj formi virtualni adresni prostor programa podeljen je na mnoštvo nezavisnih adresnih prostora koji se nazivaju **segmenti** (engl. **segments**).

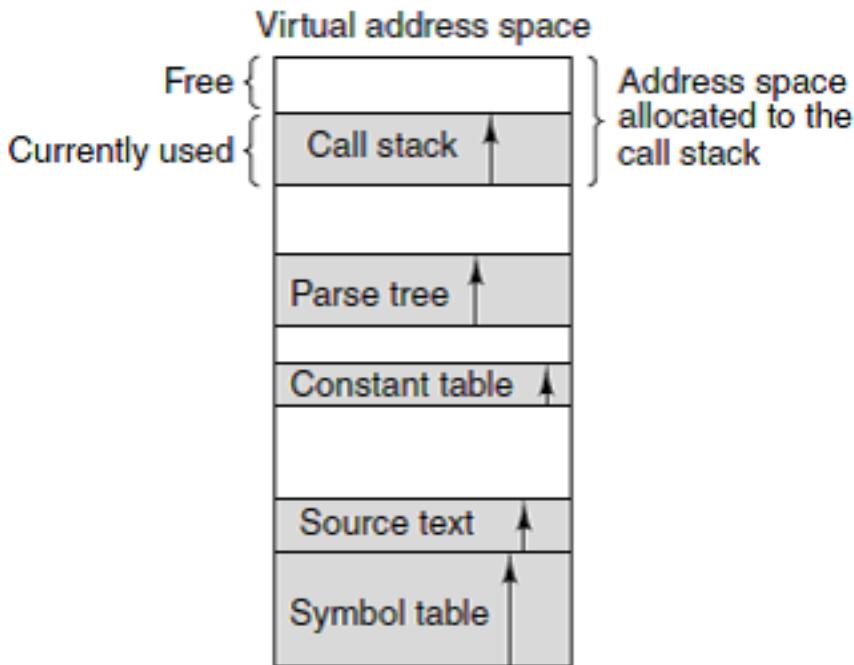
Uočavamo osnovnu razliku u odnosu na straničenje gdje je virtualni adresni prostor jednodimenzionalni niz koji ide od nulte adrese do nekog maksimuma, pri čemu adrese slike jedna za drugom.

Na primer programski prevodilac tokom prevođenja može da napravi brojne tabele kao što su:

- Tabelu simbola za imena i atributе promenljivih.
- Izvorni tekst sačuvan za potrebe štampanja listinga.
- Tabelu sa svim korišćenim konstantama, kako celobrojnim tako i onim u formatu pokretnog zareza.
- Stablo koje sadrži sintaksnu analizu programa.
- Stek za pozive koje procedurama upućuje programski prevodilac.

Svaka od prvih četiri tabela raste tokom prevođenja programa. Poslednja raste i smanjuje se na nepredvidiv način.

U jednodimenzionalnoj memoriji, za ovih pet tabela morala bi se odvojiti susedna područja adresnog prostora, kao na Slici 1.



Slika 3.1 U jednodimenzionalnom adresnom prostoru s rastućom tabelom, jedna tabela može ući u prostor one druge [Izvor: Autor]

SEGMENTI

Segmenti čine potpuno nezavisni adresni prostor

Svaki segment se sastoji od linearog niza adresa, počevši od 0, pa do nekog maksimuma.

Dužina svakog segmenta može da bude između 0 i maksimalno dozvoljene dužine.

Različiti segmenti mogu da budu, a obično i jesu, različite dužine.

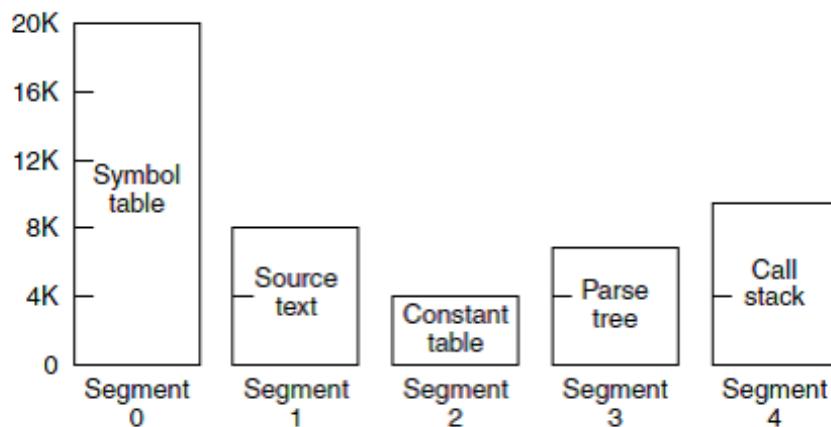
Dužina segmenta za stek može se povećavati kad god se nešto stavi na njega i smanjivati kad god se nešto s njega uzme.

Pošto svaki segment predstavlja zaseban adresni prostor, različiti segmenti mogu da menjaju veličinu nezavisno jedan od drugog.

Ako je steku u određenom segmentu potrebno više adresnog prostora, on ga može dobiti jer u njegovom adresnom prostoru ema ničega sa čim bi se sudario. Naravno, i segment se može potpuno popuniti, ali su segmenti po pravilu veoma veliki, pa se to retko događa.

Da bi pristupio adresi u ovakvoj, segmentiranoj ili dvodimenzionalnoj memoriji, program mora da zada adresu iz dva dela:

1. broj segmenta i
2. adresu unutar segmenta.



Slika 3.2 Segmentirana memorija dozvoljava svakoj tabeli da raste ili se smanjuje nezavisno od drugih tabela [Izvor: Autor]

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

IMPLEMENTACIJA SEGMENTIRANJA

Segmentiranje se može implementirati na jedan od dva načina: razmenjivanjem segmenata iz memorije i diska i straničenjem.

U prvom slučaju, u svakom trenutku u memoriji postoji određen skup segmenata.

Ako se referencira segment koji trenutno nije u memoriji, taj segment se učitava sa diska.

Ukoliko za njega nema mesta u memoriji, prethodno se na disk mora upisati jedan ili više segmenata iz memorije (osim ako su čisti, kada se samo odbacuju).

Razmenjivanje segmenata je u izvesnom smislu slično straničenju na zahtev: segmenti se u memoriju učitavaju i iz nje brišu po potrebi.

Međutim pri implementaciji, između segmentiranja i straničenja postoji bitna razlika: stranice su flksne veličine dok segmenti nisu.

Prednost segmentacije u odnosu na ostale koncepte se ogleda u zaštiti memorije.

Dodeljivanje memorije kod segmentacije je slično kao i kod straničenja. Jedina razlika je u tome što su segmenti najčešće različite veličine.

Dodela memorije predstavlja problem dinamičke dodele memorije i može se rešiti korišćenjem sličnih algoritama kao i kada je straničenje u pitanju.

IMPLEMENTACIJA SEGMENTIRANJA,

Segmenti predstavljaju logički definisane delove programa.

S obzirom da segmenti predstavljaju logički definisane delove programa, verovatno je da de se sve stavke segmenta koristiti na isti način.

Dakle, očekivano je da neki segmenti sadrže instrukcije, a neki podatke.

Segmenti koji sadrže instrukcije mogu se označiti tako da se mogu samo čitati i da se ne mogu modifikovati, čime se štite od neželjenih promena i pristupa ako se instrukcija ne može samo modifikovati.

Veličina segmenata može biti različita, pa zato segmentacija podseća na dinamičko particonisanje.

Razlika je u tome što se kod segmentacije proces ne nalazi u jednom memorijskom bloku, već se delovi procesa mogu naći bilo gde u memoriji.

U nekim implementacijama kombinuju se segmentacija i straničenje kako bi se iskoristile prednosti i umanjili nedostaci koje imaju ova dva koncepta.

Segmentacija sa straničenjem predstavlja pristup koji podrazumeva podelu segmenata na stranice čime se omogućava da segmenti ne moraju da se smeštaju u neprekidne memorijske blokove.

▼ Poglavlje 4

Virtuelna memorija Core i7

VIRTUELNA MEMORIJA I7 PROCESORA

Core i7 ima složen sistem virtuelne memorije koji podržava straničenje na zahtev čisto segmentiranje i segmentiranje uz straničenje.

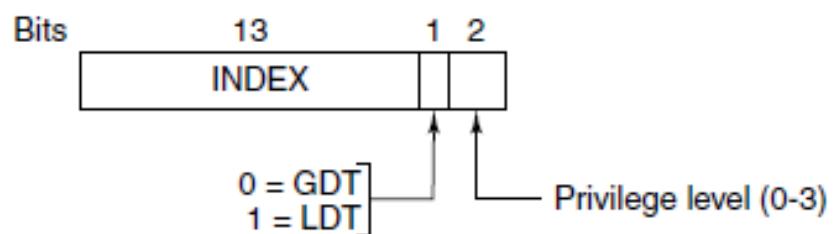
Virtuelnu memoriju i7 procesora čine dve tabele: **tabela lokalnih deskriptora** (engl. Local Descriptor Table, **LDT**), i

tabela globalnih deskriptora (engl. Global Descriptor Table, **GDT**).

Svaki program ima sopstvenu **LDT** tabelu ali postoji samo jedna **GDT** tabela koju dele svi programi u istom računaru.

U **LDT** tabeli navode se segmenti koji su lokalni za određeni program, uključujući i segmente sa kodom, podacima, stekom i dr. dok su u **GDT** tabeli sistemski segmenti, uključujući i segmente sa operativnim sistemom.

Program koji se izvršava na Core i7 procesoru, pre nego što pristupi segmentu, mora da učita selektor za taj segment u jedan od registara za segmente.



Slika 4.1 Selektor na Core i7 [Izvor: Autor]

VIRTUELNA MEMORIJA COREI7 PROCESORA

Virtuelna memorija Core i7

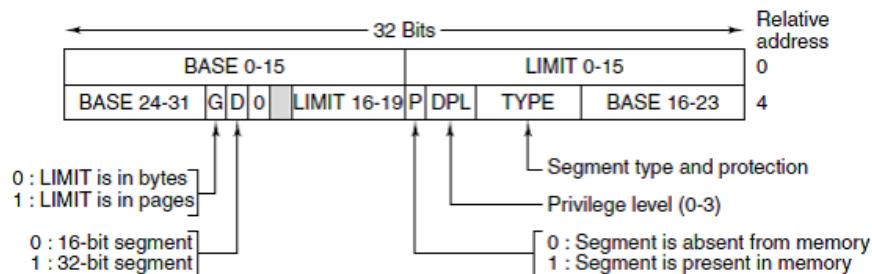
Jednim od bitova selektora saopštava se da li je segment lokalni ili globalan (tj. da li se nalazi u tabeli **LDT** ili u tabeli **GDT**).

Drugih 13 bitova su broj odrednica u **LDT** ili **GDT** tabeli, tako da svaka od tih tabela može da čuva najviše 8 KB (2^{13}) deskriptora segmenata. Preostala 2 bita služe za zaštitu i njih ćemo objasniti kasnije.

Deskriptor 0 je nevažeći ako se koristi, program upada u klopu. On se može bezbedno učitati u registar za segmente i tada označava da registar za segmente nije trenutno raspoloživ, ali ako se upotrebi, program završava u klopcu.

U trenutku dok se selektor učitava u registar za segmente, iz **LDT** ili **GDT** tabele preuzima se odgovarajući deskriptor i smešta u interne registre MMU jedinice, tako da je lako dostupan.

Deskriptor se sastoji od 8 bajtova koji sadrže osnovnu adresu segmenta, njegovu veličinu i druge informacije (Slika2).



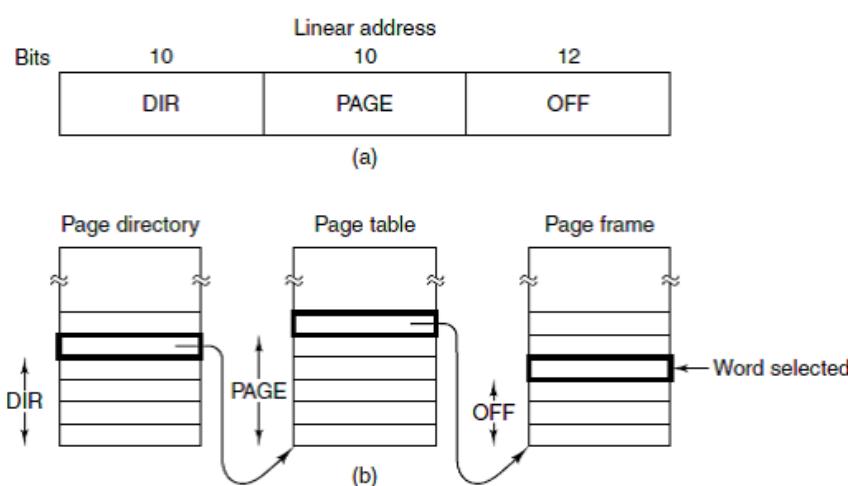
Slika 4.2 Deskriptor segmenta sa kodom na i7 [Izvor: Autor]

KATALOG STRANICA CORE I7 PROCESORA

Svaki aktivan program ima katalog stranica sa 1024 odrednica od po 32 bita

Katalog stranica se smešta na adresu na koju ukazuje globalni registar. Svaka odrednica iz ovog direktorijuma ukazuje na tabelu stranica koja takođe sadrži 1024 odrednica sa po 32 bita.

Odrednice tabele stranica ukazuju na okvire za stranice.



Slika 4.3 Preslikavanje linearne adrese u fizičke adrese [Izvor: Autor]

Na Slici 3. prikazana je linearна адреса подељена у три поља:

DIR.

PAGE i

OFF.

Prvo se koristi поље **DIR** као индекс за каталог страница да би се пронашao показиваč на одговарајућу табелу страница. Затим се користи поље **PAGE** као индекс за табелу страница да би се пронашla физичка адреса оквира за странице. На крају се поље **OFF** додајe на адресу оквира странице да би се добила физичка адреса адресираног байта или реци.

Одредnice табеле странице имају по 32 бита, од чега 20 битова садрže број оквира за странице. Међу преосталим битовима су бит за приступ и "лоš" бит које хардвер поставља да би ih користио оперативни систем, битови за заштиту и други слични битови.

ZAŠTITA NA CORE I7 PROCESORU

Core i7 процесор подржава 4 нивоа заштите.

Tema заштите је уско повезана са виртуелном меморијом.

Core i7 процесор подржава четири нивоа заштите.

Активни програм је у сваком моменту на неком од нивоа заштите који одговара двобитном пољу унутар njegovog регистра **PSW** (engl. **Program Status Word**), hardverskog регистра који чува кодove uslova i razne druge statusne bitove.

Такођe сваки сегмент система има сопствени ниво заштите.

Све док програм користи искључivo сегменте на сопственом нивоу, све тече у redu. Покушаји да се приступи подацима на виšem нивоу dozvoljavaju se. Покушаји да се приступи подацима на nižem нивоу nelegalni su i hvataju se u "klopku". Покушаји да се pozovu procedure на drugačijem нивоу (виšem ili nižem) dopušteni su, ali na brižljivo kontrolisan način.

Da biste ostvarili poziv između različitih нивоа, инструкција **CALL** мора да садржи selektor umesto адресе. Овaj selektor ukazuje на дескриптор зван **prolaz za pozive** (engl. **call gate**), он дaje адресу procedure која се poziva. На тaj начин nije могућe skočiti usred proizvoljnog сегмента koda koji se nalazi na неком другом нивоу. Njemu se može приступити само preko zvaničnih улазних тачака.

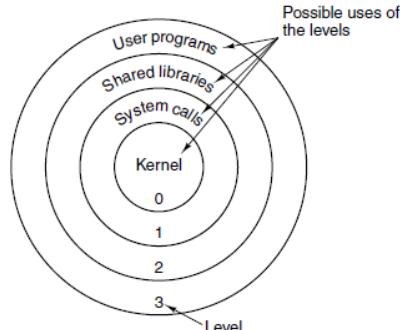
На Slici 4. приказана је примена ovог механизма.

На нивоу 0 налазимо језгро оперативног система које рукује улазно-излазним операцијама, ради с меморијом и с другим важним процесима.

На нивоу 1 налази се потпрограм за obradu системских poziva. Korisnički programi mogu pozivanjem procedura aktivirati sistemske pozive, ali se može pozivati само određen zaštićen broj procedura.

Na nivou 2 nalaze se biblioteke procedura njih možda dele mnogi programi koji se uporedo izvršavaju. Korisnički programi mogu da pozivaju ove procedure, ali ne smeju da ih menjaju.

Na kraju, na nivou 3 izvršavaju se korisnički programi koji imaju najmanju zaštitu.



Slika 4.4 Zaštita na Core i7 procesoru [Izvor: Autor]

▼ Poglavlje 5

Hardverska virtuelizacija

VIRTUELIZACIJA

Virtuelizacija je simulacija softvera ili hardvera na kome drugi softver radi.

Upotrebom ove tehnologije u domenu virtuelizacije serverskih i klijentskih operativnih sistema, jednostavno rečeno, postižemo da više operativnih sistema radi u paraleli na istoj mašini.

Ovim pristupom značajno je olakšana administracija, jer se sve svodi na administraciju jedne mašine.

Više nema potrebe upravljati se starim modelom „**jedan server - jedna aplikacija**“.

Moguće je imati više servera sa različitim operativnim sistemima, tako da se svi pokreću na istoj hardverskoj platformi. Uštede i prednosti ovakvog setupa su očigledne.

Svaki server se može posmatrati kao posebni entitet, tj. kao posebna mašina.

Otkaz jednog takvog entiteta nema uticaja na rad glavne (**host**) mašine, platforme za virtuelizaciju, niti na rad ostalih entiteta sa kojima deli resurse.

Sve ovo daje jedan veliki plus virtuelizaciji u segmentu pouzdanosti.

Dovoljno je uložiti sredstva u pouzdanu hardversku konfiguraciju i osigurati redundansu i bezbednost podataka na njoj.

Rešenja za virtuelizaciju po pravilu omogućuju relativno lako dodavanje novih servera i premeštanje podataka sa jedne na drugu **host mašinu**, što je dodatni plus ove tehnologije u domenu skalabilnosti.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

HARDVERSKA VIRTUELIZACIJA,

Hardverska virtuelizacija je najrasprostranjeniji i najpopularniji vid virtuelizacije.

Hardverska virtuelizacija predstavlja virtuelizaciju računara ili operativnih sistema.

Postoji nekoliko bitnih termina koji se vezuju za hardversku virtuelizaciju. Koncept tehnologije se zasniva na korišćenju virtualnih mašina (engl. **virtual machine**).

Softver koji kontroliše virtuelizaciju se često naziva **hipervizor** (engl: **hypervisor**) ili virtual machine monitor.

Proces kreiranja i upravljanja virtualnim mašinama naziva se i serverska virtuelizacija prema najzastupljenijem vidu upotrebe u profesionalnim IT okruženjima.

Fizička mašina na kojoj se primenjuje virtuelizacija naziva se host mašina (eng. host - domaćin).

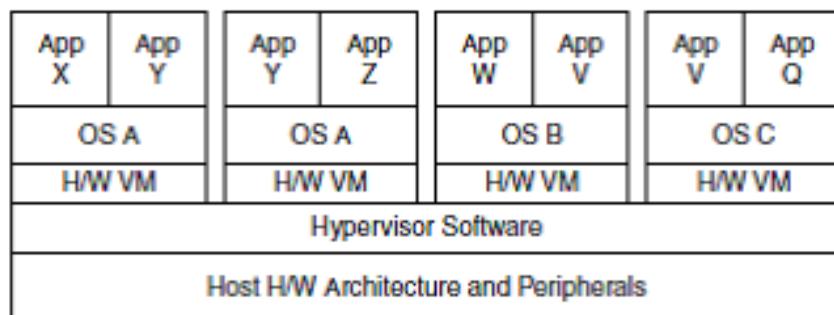
Već smo naveli da se korišćenjem ove tehnologije na jednom računaru može paralelno pokretati više operativnih sistema.

Operativni sistem koji je instaliran na host mašini i na kome se izvršava softver za virtuelizaciju naziva se host operativni sistem.

Operativni sistem koji se izvršava na virtualnoj mašini naziva se guest operativni sistem (engl. guest - gost).

Na **host** operativnom sistemu se kreira simulirano kompjutersko okruženje, odnosno virtualna mašina.

Guest operativni sistemi koriste virtualne fizičke resurse koje obezbeđuje hipervizor.



Slika 5.1 Virtuelizacija hardvera omogućava pokretanje više operativnih sistema istovremeno na istom hardveru domaćina. Hipervisor primenjuje deljenjem memorija domaćina i U/I uređaji. [Izvor:Autor]

HIPERVIZOR SLOJ

Hipervisor je sloj ili spona između fizičkih resursa host maštine i virtuelne guest maštine.

Hipervisor čini nivo ili vezu između fizičkih resursa **host maštine** i **virtuelne guest maštine**.

Aplikacije koje se izvršavaju na guest operativnom sistemu nisu limitirane host operativnim sistemom.

Guest operativni sistem se izvršava na isti način kao što bi se izvršavao na fizičkoj mašini, a ovaj sistem posmatra virtualne resurse kao fizičke resurse.

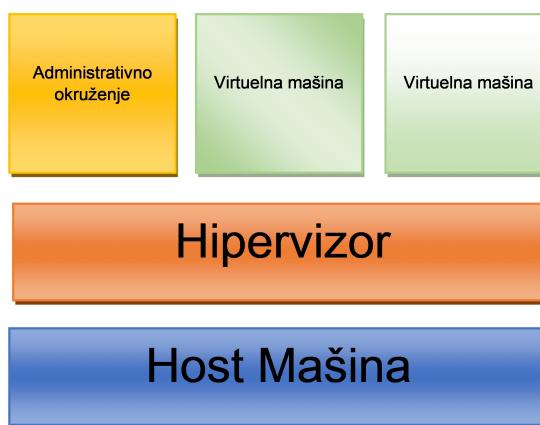
Postoji nekoliko ograničenja u smislu pristupa sistemskim resursima i perifernim uređajima na šta se može uticati konfigurisanjima u okviru virtuelne maštine.

Hipervizor može da se podeli u dve kategorije:

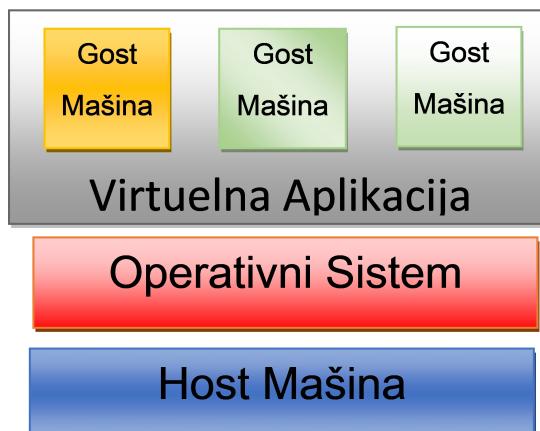
Tip 1 i Tip 2 .

Tip 1 se instalira direktno na hardver tj. slično kao kod instalacije operativnog sistema. Ovo rezultira boljim performansama u odnosu na Tip 2.

Tip 2 se instalira na postojeći operativni sistem.



Slika 5.2 Hipervizor Tip 1 [Izvor: Autor]



Slika 5.3 Hipervizor Tip 2 [Izvor: Autor]

HARDVERSKA VIZUELIZACIJA CORE I7

Virtuelizacija hardvera na i7 procesoru

Virtuelizacija hardvera na Core i7 je podržana ekstenzijama virtualnih mašina (engl: **virtual machine extensions** - **VMX**), kombinacijom ekstenzija instrukcija, memorije i prekida koje omogućavaju efikasno upravljanje virtualnim mašinama.

Sa VMX-om, virtuelizacija memorije je implementirana sa **Tabela proširenih stranica** (engl: **Extended Page Table** - **EPT**) sistemom koji je omogućen hardverskom virtuelizacijom.

EPT prevodi adrese fizičkih stranica virtualne mašine na fizičke adrese domaćina. **EPT** implementira ovo mapiranje sa dodatnom strukturom tabele stranica na više nivoa koja se prelazi tokom prevoda (engl: **Translation Lookaside Buffer** - **TLB**) propusta virtualne mašine.

Hipervizor održava ovu tabelu i na taj način može da primeni bilo koju politiku deljenja fizičke memorije.

Virtuelizacija I/O operacija, i za memorijsko mapirane I/O i I/O instrukcije, implementira se kroz proširenu podršku za prekide definisanu u **Struktura kontrole virtualne mašine** (engl: **Virtual-MachineControlStructure** - **VMCS**).

Prekid hipervizora se poziva svaki put kada virtualna mašina pristupi I/O uređaju.
Kada hipervizor primi prekid, on može implementirati I/O operaciju u softver koristeći politike neophodne da bi se omogućilo deljenje I/O uređaja među virtualnim mašinama.

✓ Poglavlje 6

Pokazne vežbe

PRIMER 1: STRANIČENJE

Primeri straničenja - Primer #1 (10 min)

ZADATAK (10 minuta):

Data je tabela stranica jednog procesa na sistemu.

Veličina stranice je 2 kB.

Koje fizičke adrese odgovaraju sledećim logičkim adresama:

1. 251
2. 3129
3. 10000
4. 6066

Stranica	Okvir
0	1
1	4
2	3
3	7
4	12

Slika 6.1 Stranica i okvir [Izvor: Autor]

Broj stranice = ceo broj količnika logičke adrese i veličine stranice. Offset = ostatak pri deljenju logičke adrese i veličine stranice. Fizička adresa = okvir * veličina stranice + offset

REŠENJE:

1) Logička adresa 251

Broj stranice = $[251/2048] = 0 \rightarrow$ okvir = 1,
Offset = $251 \% 2048 = 251,$

Fizička adresa = $1 \times 2048 + 251 = \text{2299}$

2) Logička adresa 3129

Broj stranice = $[3129/2048] = 1 \rightarrow$ okvir = 4,
Offset = $3129 \% 2048 = 1081,$

Fizička adresa = $4 \times 2048 + 1081 = \text{9273}$

3) Logička adresa 10000

Broj stranice = $[10000/2048] = 4 \rightarrow$ okvir = 12,

Offset = $10000 \% 2048 = 1808$,

Fizička adresa = $12 \times 2048 + 1808 = \text{26384}$

4) Logička adresa 6066

Broj stranice = $[6066/2048] = 2 \rightarrow$ okvir = 3,

Offset = $6066 \% 2048 = 1970$,

Fizička adresa = $3 \times 2048 + 1970 = \text{8114}$

PRIMER 2: SEGMENTACIJA

Primeri segmentacije - Primer #2 (10 min)

ZADATAK (10 minuta):

Data je sledeća tabela segmenata.

Koje fizičke adrese odgovaraju sledećim logičkim adresama:

(a) 0, 430

(b) 1, 10

(c) 2, 500

Fizičke adrese segmenta:

Početak = baza

Kraj = baza + dužina

Segment	Baza	Dužina
0	219	600
1	2300	14
2	90	100

Slika 6.2 Fizička adresa segmenta [Izvor: Autor]

Logička adresa u segmentu mora biti kraća od dužine segmenta ili dolazi do greške prekoračenja.

a) 0, 430

Fizičke adrese nultog segmenta:

Početak = 219

Kraj = $219 + 600 = 819$

Fizička adresa logičke adrese 430:

219 + 430 = 649 ✓

b) 1, 10

Fizičke adrese prvog segmenta:

Početak = 2300

Kraj = $2300 + 14 = 2314$

Fizička adresa logičke adrese 10:

2300 + 10 = 2310 ✓

c) 2, 500

Fizičke adrese drugog segmenta:

Početak = 90

Kraj = $90 + 100 = 190$

Fizička adresa logičke adrese 500:

90 + 500 = 590 ✗ Prekoračenje!

PRIMER 3: SEGMENTACIJA

Primeri segmentacije - Primer #3 (10 min)

ZADATAK (10 minuta):

Data je sledeća tabela segmenata.

Koje fizičke adrese odgovaraju sledećim logičkim adresama:

(a) 3, 400

(b) 4, 112

Fizičke adrese segmenta:

Početak = baza

Kraj = baza + dužina

Segment	Baza	Dužina
3	1327	580
4	1952	96

Slika 6.3 Fizička adresa segmenta [Izvor: Autor]

Logička adresa u segmentu mora biti kraća od dužine segmenta ili dolazi do greške prekoračenja.

a) 3, 400

Fizičke adrese trećeg segmenta:

Početak = 1327

Kraj = $1327 + 580 = 1907$

Fizička adresa logičke adrese 400:

$1327 + 400 = 1727 ✓$

b) 4, 112

Fizičke adrese trećeg segmenta:

Početak = 1952

Kraj = $1952 + 96 = 2048$

Fizička adresa logičke adrese 400:

$1952 + 112 = 2064 \times \text{Prekoračenje!}$

PROJEKTNI ZADATAK IZ PREDMETA CS120

Predmet CS120 od predispitnih obaveza sadrži i PZ koji se sastoji od tri mini-projekta.

Na predmetu CS120 - Organizacija računara svaki student samostalno radi projektni zadatak (PZ), koji se sastoji od tri mini-projekta.

Mini-projekti brane se u petoj, desetoj i petnaestoj nedelji semestra!

Svaki student dužan je da tok izrade projekta izveštava svake nedelje u obliku kratkih izveštaja (do jednog pasusa) preko LAMS lekcija u okviru dodatnih aktivnosti interaktivnih lekcija.

O formatu mini-projekata svi studenti (tradicionalni i Internet) biće obavešteni mejлом od strane predmetnog profesora/asistenta.

✓ Poglavlje 7

Zadaci za samostalni rad

ZADACI ZA SAMOSTALNI RAD 1,

Dodatni primeri za samostalno vežbanje Overlay #1 (10 minuta)

ZADATAK (10 minuta)

Glavni program (GP, veličine 60 kB) poziva u datom redosledu sledeće segmente programa:

A (20 kB),

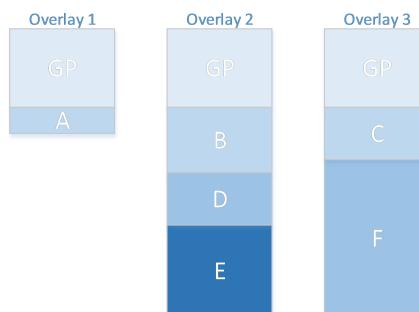
B (50 kB),

C (40 kB).

Segment B u jednom trenutku poziva segment D (40 kB), koji opet poziva segment E (70 kB).

Segment C u jednom trenutku poziva segment F (120 kB).

a) Skicirati kako se za dati primer može primeniti tehnika preklapanja (overlay)?



Slika 7.1 Overlay 1,2 i 3 [Izvor: Autor]

overlay 1: $GP+A = 60K+20K = 80K$

overlay 2: $GP+B+D+E = 60+50+40+70 = 220K$

overlay 3: $GP+C+F = 60+40+120 = 220K$

ZADACI ZA SAMOSTALNI RAD 2,

Dodatni primeri za samostalno vežbanje Overlay #2 (10 minuta)

ZADATAK (10 minuta)

Glavni program (GP, veličine 60 kB) poziva u datom redosledu sledeće segmente programa:

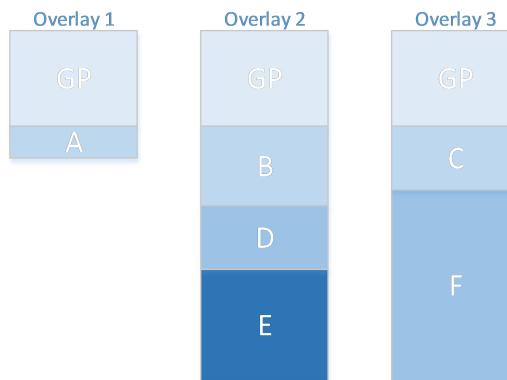
A (20 kB),

B (50 kB),

C (40 kB).

Segment B u jednom trenutku poziva segment D (40 kB), koji opet poziva segment E (70 kB). Segment C u jednom trenutku poziva segment F (120 kB).

a) Koliko je najmanje radne memorije potrebno za izvršavanje ovog programa?



Slika 7.2 Overlay 1,2 i 3 [Izvor: Autor]

$$\max(80, 220, 220) = 220 \text{ kB}$$

ZADACI ZA SAMOSTALNI RAD 3,

Dodatni primeri za samostalno vežbanje Overlay #3 (10 minuta)

ZADATAK (10 minuta)

Glavni program (GP, veličine 60 kB) poziva u datom redosledu sledeće segmente programa:

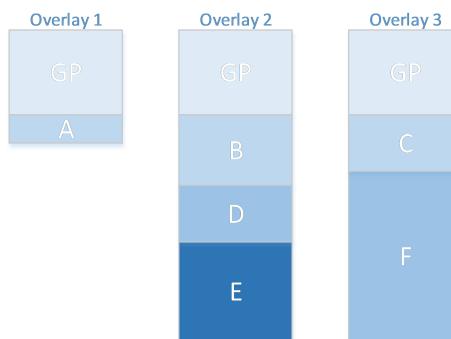
A (20 kB),

B (50 kB),

C (40 kB).

Segment B u jednom trenutku poziva segment D (40 kB), koji opet poziva segment E (70 kB). Segment C u jednom trenutku poziva segment F (120 kB).

a) Koliko bi memorije bilo potrebno bez primene tehnike preklapanja?



Slika 7.3 Overlay 1,2 i 3 [Izvor: Autor]

$$GP + A + B + C + D + E + F =$$

$$60 + 20 + 50 + 40 + 70 + 40 + 120 = 400 \text{ kB}$$

✓ Poglavlje 8

Domaći zadatak

DOMAĆI ZADATAK #10

Izrada domaćeg zadatka #10 okvirno traje 20 minuta.

Domaći zadatak #10, zadatak #1 (20 minuta)

Glavni program (GP, veličine 40 kB) poziva u datom redosledu sledeće segmente programa: A (kB), B (kB), C (kB), Segment B (kB) u jednom trenutku poziva segment D (kB), koji opet poziva segment E (kB). Segment C u jednom trenutku poziva segment F (kB).

- a) **Skicirati kako se za dati primer može primeniti tehnika preklapanja (overlay)?**
- b) **Koliko je najmanje radne memorije potrebno za izvršavanje ovog programa?**
- c) **Koliko bi memorije bilo potrebno bez primene tehnike preklapanja?**

- A ima vrednost poslednja cifra broj_indeksa *10
 - B ima vrednost predposlednja cifra broj_indeksa *20
 - C ima vrednost poslednja cifra broj_indeksa *20
 - D ima vrednost predposlednja cifra broj_indeksa *10
 - E ima vrednost poslednje dve cifre broj_indeks % 7*10
 - F ima vrednost poslednje dve cifre broj_indeks % 10*10
- U slučaju da je vrednost modulo 0 dodaje se broj 1 Primer:
- **Za indeks br: 4356; A=60; B=50; C=120; D=100; E=10; F=6**

Predaja domaćeg zadatka

Tradicionalni studenti:

Domaći zadatak treba dostaviti najkasnije 7 dana nakon vežbi, za 100% poena. Nakon toga poeni se umanjuju za 50%.

Domaći zadatak poslati predmetnim asistentima, sa predmetnim profesorima u CC.

Predati domaći zadatak koristeći .doc/.docx uputstvo dato u prvoj lekciji.

Internet studenti treba poslati domaće zadatke najkasnije do 10 dana pred izlazak na ispit predmetnom asistentu zaduženog za internet studente.

Napomena:

Svaki domaći zadatak treba da bude napisan prema dokumentu za predaju domaćih zadataka koji je dat na kraju interaktivne lekcije.

▼ Poglavlje 9

Zaključak

ZAKLJUČAK

Rezime lekcije #10 - Sloj operativnog sistema

U lekciji #10 najpre je bilo reči o Sloju operativnog sistema

Operativni sistem se može posmatrati kao tumač za određene arhitektonske karakteristike koje se ne nalaze na ISA nivou. Glavne među njima su virtuelna memorija, virtuelne I/O instrukcije i sredstva za paralelnu obradu.

Virtuelna memorija je arhitektonska karakteristika čija je svrha da omogući programima da koriste više adresnog prostora nego što mašina ima fizičke memorije, ili da obezbedi konzistentan i fleksibilan mehanizam za zaštitu i deljenje memorije.

Može se implementirati kao čisto stranice, čista segmentacija ili kombinacija ova dva. U čistom stranicaju, adresni prostor se deli na virtuelne stranice jednake veličine. Neki od njih su mapirani u fizičke okvire stranica. Drugi nisu mapirani.

MMU prevodi referencu na mapiranu stranicu u ispravnu fizičku adresu. Referenca na nemapiranu stranicu izaziva grešku stranice.

Konačno, predstavljena je virtuelna memorija kao i hardverska virtuelizacija na Core i7 procesoru.

Literatura:

1. A. Tanenbaum, Structured Computer Organization, Chapter 06, pp. 437 – 464