Operativni sustav - upravljanje procesima

Obadić, Nikolina

Sveučilište Jurja Dobrile u Puli
Odjel za informacijsko-komunikacijske tehnologija

NIKOLINA OBADIĆ

OPERATIVNI SUSTAV: UPRAVLJANJE PROCESIMA

Završni rad

Pula, _____, 2017 godine

Sveučilište Jurja Dobrile u Puli

Odjel za informacijsko-komunikacijske tehnologija

NIKOLINA OBADIĆ

OPERATIVNI SUSTAV: UPRAVLJANJE PROCESIMA

Završni rad

JMBAG: 0303046010, redoviti student

Studijski smjer: Informatika

Predmet: Osnove IKT

Mentor: prof. dr. sc.Vanja Bevanda

Pula, rujan, 2017 godine.

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani Nikolina Obadić, kandidat za prvostupnika informacijsko-komunikacijske tehnologije ovime izjavljujem da je ovaj Završni rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Završnog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

_____

U Puli, rujan, 2017 godine

IZJAVA

o korištenju autorskog djela

Ja, Nikolina Obadić dajem odobrenje Sveučilištu Jurja Dobrile
u Puli, kao nositelju prava iskorištavanja, da moj završni rad pod nazivom „Operativni sustav-upravljanje procesima“ koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.
Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli, rujan, 2017 godine.

Potpis

_____

Sadržaj

Uvod

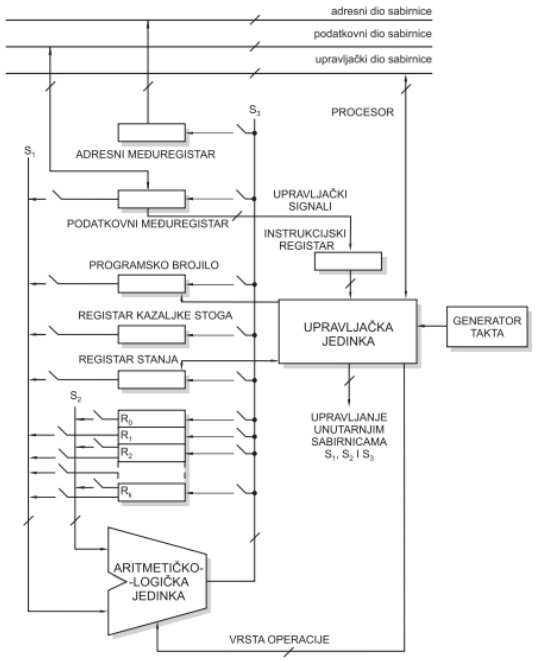
Tijekom godina razvoja računala i infomacijskih tehnologija, računalo i popratna programska podrška svakim danom sve je više bila dio naših života te je danas rad bez računala i potpore računalnog sustava gotovo nezamisliv, a svi poslovi koji se obavljaju gotovo su neostvarivi bez istog. Korisnik računala već se prilikom prvog korištenja susreće sa operativnom sustavom (OS). OS je dio softvera. Softver je jedna od dviju glavnih komponenti računala, koji se definira kao skup programa koji upravljaju radom računala. Druga glavna komponeta je hardver koji se može opisati kao opipljivi, fizički dio računala s kojim upravlja softver. U ovom radu fokus će biti na OS koji je podsustav sistemskog softvera, a neophodan je za rad na računalu na taj način da upravlja cjelokupnim djelovanjem računala. Sistemski softver osim na OS dijeli se još na pogonske programe koji omogućuju komunikaciju OS-a sa uređajima, pomoćne i uslužne programe koji pomažu OS-u u izvođenju specijaliziranih poslova te na alate za razvoj i izvođenje drugih programa kao što su prevoditelji, pretraživači, programski jezici i drugo. Kada se govori o operativnom sustavu, najjednostavnije se opiše kao skup programa koji omogućuje efikasno upravljanje hardverom i djeluje kao posrednik između hardvera i korisnika računala. Svaki OS mora biti efikasan ili praktičan, a nekim slučajevima i oboje. Koncept operativnog sustava danas je takav da podržava od najjednostavnijih aplikacija do kompliciranih poslovnih aplikacija i sve ih izvodi bez poteškoća. U ovom radu cilj je pojasniti upravljanje procesima u operativnom sustavu gdje će se pobliže pojasniti višedretveni radi na jednoprocesorskom i višeprocesorskom sustavu i svi problemi koji mogu nastati tijekom izvođenja dretvi te moguća rješenja tih problema. Struktura rada sastoji se od devet poglavlja koja su povezana sa upravljanje procesima. Prvo poglavlje započinje kratkim uvodom u računalni sustavom u kojem je opisano računalo sa najvažnijim dijelovima koji su usko povezani za OS sustav. U drugom poglavlju nastavlja se na glavne funkcije OS. Treće poglavlje opisuje upravljanje procesima te višedretvene radove na jedno i višerocesorskom sustavu. Slijedeće, četvrto poglavlje govori o međusobnom isključivanju u sustavu sa više dretvi te kako se ostvaruju zadaci i podzadaci u takvom sustavu. U petom poglavlju objašnjeno je međusobno isključivanje kod dviju dretvi te pokušaji međusobonog isključivanja dok je u šestom poglavlju objašnjen Lamportov algoritam i sklopovska potpora kod međusobnog isključivanja. Sedmo poglavlje odnosi se na potpuni zastoj kod dretvi. Osmo poglavlje odnosi se na jezgru operativnog sustava u kojoj se javljaju razna stanja dretvi. Te na kraju ovog rada u zadnjem, devetom poglavlju govori se o ostvarivanju jezgri u višeprocesorskom sustavu.

1. Računalni sustav

Računalni sustav sastoji se od procesora, radnog spremnika, vanjskih spremnika i različitih ulazno-izlaznih jedinica. Te komponente zajedno čine sklopovlje računala. Osnovne funkcije računala su ulaz, obrada, izlaz, pohrana i upravljanje. Današnja računala temelje se na Jonh von Neumannovom modelu iz 1945. godine. Njegov model sastoji se od dijelova koji su:

- Ulazni dio – unos podataka i instrukcija programa iz okoline
- Izlazni dio- prijenos rezultata programa u okolinu
- Radni (glavni) spremnik- pohranjivanje svi podataka i instrukcija programa koji se unose izvana, pohranjivanje rezultata djelovanja instrukcija
- Aritmetičko – logička jedinka- izvodi instrukcije koje su zadane aritmetičkim i logičkim operacijama
- Upravljačka jedinka[1]- dohvaća instrukcije iz spremnika, dekodira ih te upravlja aritmetičko- logičkom jedinkom, upravlja ulazno-izlaznim dijelovima.

Središnji dio svakog računala je spremnik. Njegovo ime nastalo je prema tome što kod izvođenja programa u njemu se moraju nalaziti instrukcije programa i podaci na koje te instrukcije djeluju. Kod modela računala prema von Neumannu, procesor čine skup registara, aritmetičko-logična jedinka i upravljačka jedinka.



Slika 1. Pojednostavljeni model procesora *(Izvor: Budin L., Golub M., Jakobović D., Jelenković L. : Operacijski sustavi, 3. Izdanje, Zagreb, 2013 (str. 16))*

[1] Budin L., Golub M., Jakobović D., Jelenković L. : Operacijski sustavi, 3. Izdanje, Zagreb, 2013. (str.9)

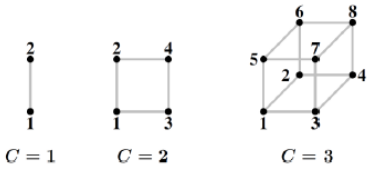
Registri služe za pohranu svih informacijskih sadržaja koji ulaze i izlaze iz procesora te se u njemu transformiraju. U nastavku su opisani pojedini registri koji se koriste u radu procesora:

- Adresni međuregistar- služi za adresiranje spremnika i ostalih dijelova računala
- Podatkovni međuregistar- posrednik kod razmjene sadržaja između procesora i ostalih dijelova računala
- Instrukcijski registar- prijenos instrukcija dobivenih iz spremnika
- Programsko brojilo- registar sa adresom instrukcije koju mora slijedeću obaviti
- Registar kazaljke stoga- služi za poseban način adresiranja spremnika u kojem se rezervira skupina uzastopnih bajtova i zapišu se u registar kazaljke stoga. Pomoću tog registra ostvaruje se pohrana podataka na stog
- Bitovi stanja registra- služi za zapisivanje različitih zastavica koje označavaju ispravnost ili neispravnost rezultata i operacija
- Opći registri- služi za pohranjivanje operandi i rezultata i priprema adrese budućih pristupa do spremnika.

1.1. Povijest razvoja procesora

Na samom početku dizajniranja procesorskih sustava bilo je mnogo problema ispred dizajnera koji su radili na pronalasku rješenja jaza između brzine registra i glavne memorije koja bi u konačnici rezultirala povećanjem brzine izvođenja samog programa. Rješenje se nalazilo u povećanju radnog takta procesora (paralelizam na razini instrukcija (instruction-level parallelism)). Takav način rada obuhvaća više mehanizama u kojima se paralelno u procesor dovode instrukcije istog programa koje se paralelno obrađuju. Ako bi se jedna prosječna instrukcija obrađivala u pet faza obrade tada bi se paralelnom obradom svih pet instrukcija prosječno vrijeme svelo na samo jedan takt. Problem koji su uočili tijekom ovakvog rada procesora bio je kod instrukcija koje narušavaju tok programa. Prvo rješenje koje su provodili bilo je uvođenje tranzistora u procesor ali su ih uvodili sve više i više i ubrzo odbacili tu ideju. Slijedeća ideja bila je da umjesto paralelne obrade instrukcija istog programa, obrađuju paralelno instrukcije različitih programa tj različitih dretvi. Takva obrada instrukcija naziva se multithreading. Niti jedna od ovih navedenih rješenja nije idealno. Dretve dijele istu priručnu memoriju, a kod multithreadinga su duplirani samo neki elementi CPU-a za razliku od višejezgrenih procesora. Iako zadani cilj za povećanjem performansi za dva puta nije do kraja zadovoljen ipak je multithreading bio koristan te se koristi i kod

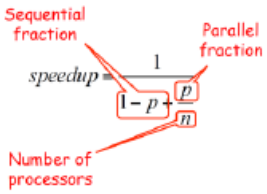
višejezgrenih procesora. Samo povećanje radnog takta procesora sa sobom je donosilo i neke posljedice u obliku povećanje potrošnje i zagrijavanje procesora pa je sljedeći korak bio razuman u obliku smještanja dvije ili više jezgri u jedan procesorski čip. Na taj način u računalo se može ugrađivati i po nekoliko višejezgrenih procesora. Postoje dvije vrste arhitektura, a to su centralna djeljiva memorija i arhitektura distribuirane djeljive memorije. Centralna djeljiva memorija označava dva procesora koja nemaju zajedničku priručnu memoriju, a do memorije dolaze na uobičajen način, dok arhitektura distribuirane djeljive memorije je djeljiva između svih procesora, određeni dijelovi memorije su bliži pojedinim procesorima.



Slika 2. Povezivanje procesora i pripadajuće memorije *(Izvor: Zlatko Sirotić-Utjecaj razvoja mikroprocesora na programiranje)*

To znači da procesor najbrže pristupa svojem lokalom dijelu glavne memorije dok brzina prema udaljenim dijelovima ovisi o udaljenosti procesora od drugog procesora kojem pripada taj dio lokalne memorije. U takvoj arhitekturi procesori su direktno povezani sa određenim brojem drugih procesora te nisu samo međusobno povezani preko memorijske sabirnice.

Općenito kod više procesorskih sustava i višejezgrenih procesora rast performansi se rijetko proporcionalno povećava sa brojem jezgri dok povećanje radnog takta kod jednoprocesorskih i jednojezgrenih sustava povećanje je skoro pa linearno. Ukoliko se proporcionalno povećanje i postigne kod višejezgrenih sustava tada takvi sustavi služe kao podrska bazi podataka i aplikacijskih servera kod kojih su pojedinačni procesi/dretve međusobno neovisni.



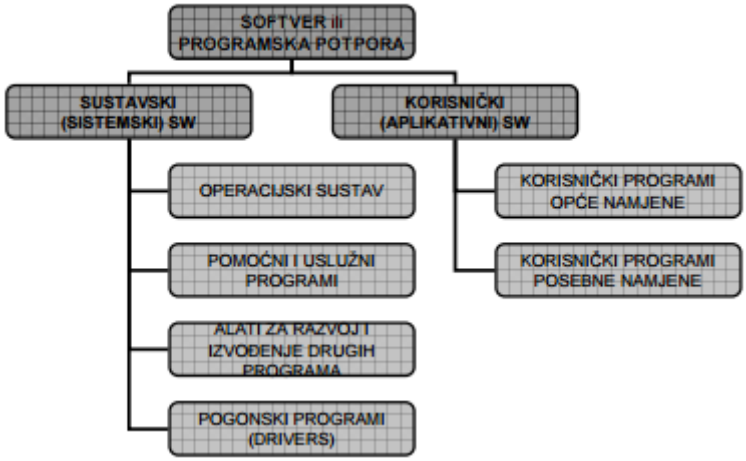
Slika 3. Amdahlov zakon *(Izvor:Zlatko Sirotić-Utjecaj razvoja mikroprocesora na programiranje)*

Amdahlov zakon pojašnjava da ako je u nekom programu proporcija onih dijelova programa koji se mogu paralelno izvršavati jednaka p koji označava da je proporcija dijelova koji se mogu paralelno izvršiti jednaka 1-p onda se povećanjem CPU-a može dobiti ovo povećanje.[2]

[2] http://www.istratech.hr/wp-content/uploads/2012/11/case2012_1.pdf- 10.06.2017.

1.2. Softver

Programi koji upravljaju radom računala i izvode obradu podataka nazivamo softverom.
Da bi se stvorio bilo kakav program potreban je programski jezik koji će omogućiti izvođenje bilo kakvih procesa na računalu. Programski jezik je umjetno stvoren jezik koji se koristi za kontrolu ponašanja nekog računala ili stroja.[3]



Slika 4. Osnovna podjela softvera *(Izvor: Veleri.hr-softver-predavanja)*

Osnova podjela softvera vrši se na:

- Sistemski- dijelovi programa koji omogućuju korisniku lakše i efikasnije korištenje resursa
- Korisnički (aplikacijski)- programi koji su namijenjeni rješavanju konkretnih problema ili zadataka

Sistemski softver još se dalje dijeli na :

- Operativni sustav (OS)- upravlja cjelokupnim djelovanjem računala
- Pogonski programi- omogućuju komunikaciju OS sa ugrađenim uređajima ili uređajima koji su povezani sa računalom
- Pomoćni i uslužni programi- pomoć OS kod izvođenja specijaliziranih instrukcija
- Alati za razvoj i izvođenje drugih programa[4]- kao što su prevoditelji, pretraživači pogrešaka, programski jezici i drugo.

Tijekom razvoja softvera prolazi se kroz nekoliko faza. Prva faza je alfa verzija koja je samo idejna verzija programa te se može koristiti. Nakon alfa verzije dolazi beta verzija u kojoj se daje na testiranje grupi korisnika, a korisnici zatim prate ponašanje programa i ocjenjuju ga. Treća faza naziva se izgled u kojoj softver ima potencijalni krajnji izgled. Zadnja faza je

[3] http://veleri.hr/~elena/Predavanja/softver.pdf , 02.07.2017.
[4] http://veleri.hr/~elena/Predavanja/softver.pdf , 28.06.2017.

finalna verzija u kojoj se službeno predstavlja program za tržište.

2. Operativni sustavi - funkcije

Danas se u svim segmentima života koriste računala različitih namjena kao što su osobna računala, pametni telefoni ili prijenosna računala i drugo. Sa svima njima upravlja operativni sustav (OS) koji se razlikuje od uređaja do uređaja na kojima se nalazi.

Osnovni elementi bilo kojeg OS u načelu su jednaki. OS se može definirati kao skup osnovnih programa koji omogućuju izvođenje radnih zahvata na računalu i omogućuju izvođenje operacija računala te pripada sismemskom softveru. OS upravlja sustavom prema zahtjevima korisnika, omogućuje korisnicima i programima jednostavno korištenje. Korisnik ne mora poznavati složenost i detalje sklopovlja koje se koristi kod izvedbe naredbi računalu.[5]

Svrha korištenja OS-a je olakšana upotreba računala, višeprogramski rad, učinkovitost sustava, oslobođenje korisnika od stalne kontrole nad računalom i drugo. Najvažnije funkcije OS-a su :

- Upravljanje procesima
- Upravljanje memorijom
- Upravljanje datotekama
- Upravljanje ulaznim i izlaznim jedinicama
- Upravljanje zaštitom
- Upravljanje sučeljem[6].

Prema broju korisnika OS se dijeli na :

- Jednokorisničke- svi resursi dostupni su samo jednom korisniku
- Višekorisničke- omogućen je rad za više korisnika istovremeno
- Mješovite- može ga koristiti više korisnika ali ne istovremeno.

Neka od glavnih obilježja OS-a su još istovremenost, djelotvornost, sigurnost, fleksibilnost, uporabljivost i djeljivost resursa. Sve navedene funkcije i obilježja očitiju se tijekom izvođenja zadataka.

[5] http://www.zemris.fer.hr/~leonardo/os/fer/OS-skripta.pdf , 28.06.2017.

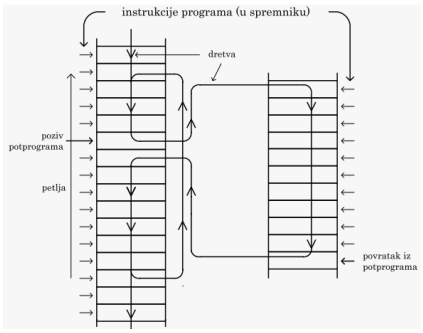
[6] Tanenbaum A. S., Bos H. - Modern operating systems, 4. izdanje, Amsterdam, 2015. (str.38), 30.07.2017.

3. Operativni sustav: upravljanje procesima

Proces se može opisati kao program koji je zapisan ili pohranjen u strojnom obliku u računalu. Sastoji se od niza instrukcija koje procesor prilikom izvođenja programa mora razumjeti i obaviti ispravno. Tijekom izvođenja programa, njemu se pripisuju neka vremenska svojstva kao što su :

- Trenutak početka izvođenja programa
- Trenutak završetka izvođenja programa
- Trajanje izvođenja programa
- Zaustavljanje izvođenja programa i drugo.

Vremenska svojstva programa daju obilježja koja postaju proces. Također takav proces naziva se još i računalni s obzirom da se takvi procesi odvijaju u računalu. Računalni proces ne obuhvaća samo instrukcije programa tijekom njihova izvođenja već i sve druge pojave koje su povezane sa izvođenjem programa kao što su strukture podataka i datoteka, ulazne i izlazne operacije i slično. Unutar svakog procesa postoji barem jedna instrukcijska dretva. Dretva se može definirati kao izvođenje niza instrukcija.



Slika 4. Instrukcija dretve

(*Izvor: Leonardo Jelenković:Operacijski sustavi- skripta 2015./2016.)*

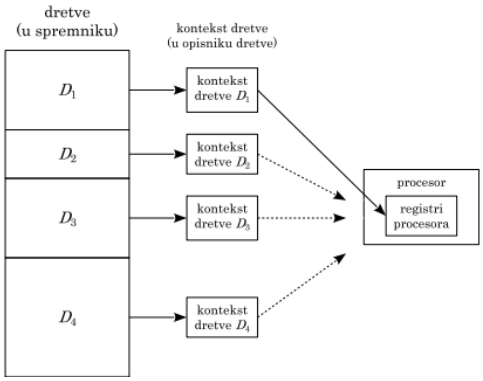
Da bi se ispravno izvodila, dretvi su potrebne instrukcije, podaci i stog koji se nalaze u spremniku te registri procesora koji se nalaze u procesoru.

3.1. Višedretveni rad na jednoprocesorskom sustavu

Višedretveni rad se može primjenjivati kod jednoprocesorskih računala tako da jedan procesor koji se nalazi u računalu naizmjenično provlači jednu po jednu dretvu. Takav način rada ne ubrzava izvođenje procesa ali se omogućava izvođenje neke dretve za vrijeme u kojem druga dretva čeka zbog nekog razloga.

U slučaju kada se odvija jednoprocesorski način rada, treba se osigurati da svaka dretva radi

sa svojim skupom registra. Na takav način se postiže da se sadržaj registra procesora one dretve čije se izvođenje želi prekinuti pohrani na mjesto koje je rezervirano u spremniku dok se u registre procesora sprema sadržaj koji pripada dretvi koja se počinje izvoditi.



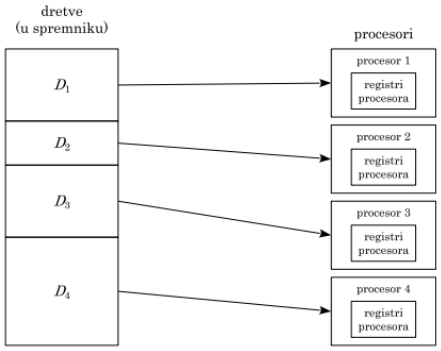
Slika 5. Višedretvenost na jednoprocesorskom sustavu (*Izvor:Leonardo Jelenković:Operacijski sustavi- skripta 2015./2016.)*

Prema slici 5, višedretvenost na jednoprocesorskom sustavu postigla se tako da se u nekom trenutku aktivna dretva zamijeni drugom. Postupak koji se izvodi tijekom zamjene jedne dretve sa drugom je:

- prekid izvođenja aktivne dretve
- spremanje konteksta aktivne dretve
- odabir nove aktivne dretve
- obnavljanje konteksta nove aktivne dretve
- nastavak rada nove aktivne dretve.

3.2. Višedretveni rad na višeprocesorskih sustava

Danas gotovo sva računala podržavaju višedretveni rad. Taj način rada omogućuje jednostavniju izvedbu primjenskih programa tako da povećavaju učinkovitost korištenja svih dostupnih resursa sustava kao što je korištenje svih procesorskih jedinki višestrukog procesora na čipu.



Slika 6. Višedretvenost na višeprocesorskom sustavu (*Izvor: Leonardo Jelenković:Operacijski sustavi -skripta 2015./2016.)*

Višedretveni rad se ostvaruje kada se u računalu nalazi dovoljno procesora tj. da se svaka dretva nalazi na svom procesoru kao što je prikazano na slici 6. Ti se procesi mogu istovremeno izvršavati, u ovom slučaju sva 4 ili bilo koja kombinacija nekih procesa. Adresni spremnik dijeli se tako da svaka dretva dobije dio adresnog prostora iz kojeg dohvaća instrukcije i podatke. Sustav može samostalno odrediti na koji način će rasporediti dretve na dostupne procesore. Prednost višedretvenog rada nad procesima je brža komunikacija između dretvi nego procesa te je jednostavnija zbog toga što se obavlja na istom adresnom prostoru. Istovremeno pristupanje podacima od strane više dretvi ponekad je nemoguće te se mora spriječiti. U takvim situacijama dretve koriste iste varijable, a takva upotreba izazvala bi greške u radu. Da bi se spriječilo takvo ponašanje, koristi se međusobno isključivanje dretvi.

4. Međusobno isključivanje u višedretvenim sustavima

Kao što je u prethodnom poglavlju definirano, dretva prestavlja izvođenje niza instrukcija. Obavljanje tih instrukcija naziva se proces. U svakom procesu mora se nalaziti najmanje jedna dretva. Odvijanje procesa obavlja se tako da procesor izvodi tu dretvu. Ukoliko su zadaci za izvođenje složeni, oni se mogu podijeliti na podzadatke kako bi se lakše savladala složenost i na taj način omogućilo bolje iskorištavanje resursa koji su dostupni. Današnji OS još se nazivaju i suvremeni zbog mehanizama koji podržavaju izvođenje procesa sa više dretvi također su i višedretveni i višezadaćni.

Tijekom prebacivanja s jedne dretve na drugu ne treba mijenjati kontekst u procesoru već je rješenje pronađeno u mehanizmu koji onemogućava prekid koji se događa automatski u trenutku kada se pojavi sklopovski prekid ili programski prekid(kada dretva obavlja operacije koje zahtjevaju više privilegija izaziva se namjerno prekid).

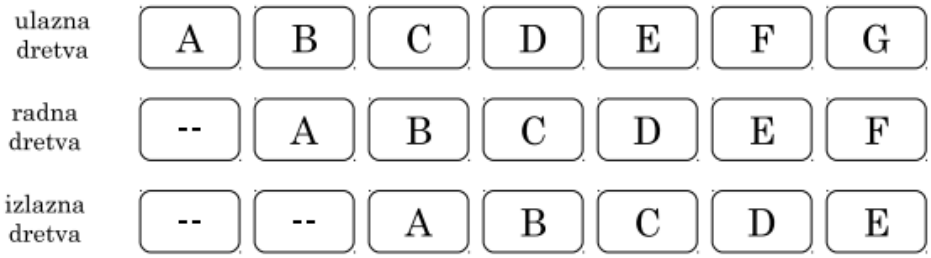
4.1. Višedretveno ostvarenje zadataka, sustav podzadataka

4.1.1. Zadaci i podzadaci

Cilj svake višedretvene organizacije je podjela podzataka na jasno razlučive.
Da bi takva podjela bila omogućena uvedena su tri podzadatka:

- podzadatak za čitanje ulaznih podataka
- podzadatak za obradu
- podzadatak za obavljanje izlaznih operacija.

Podzadaci unutar zadatka moraju se obavljati redoslijedom kojim su navedeni. Ukoliko se neki zadatak ponavlja, tada se podzadaci mogu obavljati i istovremeno. Ako bi se slijedio ovakav primjer obavljanja zadataka bilo bi dovoljno imati tri dretve. Prva je ulazna dretva kojoj je zadaća nabava podataka. Nakon što su podaci nabavljeni, prva predaje drugoj dretvi i vraća se na svoj prvotni zadatak tj. nabavu podataka. Zatim druga dretva nakon preuzimanja podataka radi obradu te predaje rezultate trećoj dretvi te kao i prva dretva vraća se na svoj glavni zadatak. Zadatak treće dretve je preuzimanje rezultata i obavljanje izlazne operacije te se također vraća na svoj početni zadatak, na preuzimanje novih rezultata.



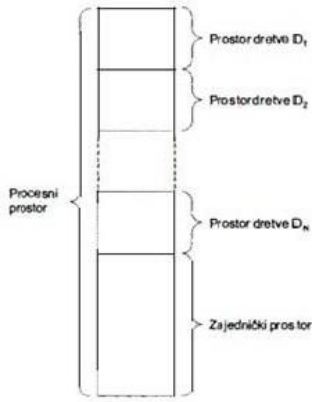
Slika 7. Cjevovodno ponašanje trodretvenog zadatka (*Izvor: Leonardo Jelenković:Operacijski sustavi- skripta 2015./2016.)*

U procesorima se koristi načelo cjevovodnog rada u kojem se preklapaju pojedine faze izvođenja uzastopnih instrukcija. Prednost kod ponašanja sklopovlja je u tome da se može namjestiti ponašanje tako da pojedine faze traju jednako, a da se izvođenje svake faze odvija u različitim dijelovima iako takvo ponašanje u gore navedenom primjeru najčešće se ne može očekivati. Da bi se moglo namjestiti takvo ponašanje bilo bi potrebno imati tri procesora od kojih bi jedan bio zadužen samo za ulaz, izlaz bi obavljao drugi procesor, dok bi treći ili centralni procesor bio zadužen za izvođenje djelomično ulazne i djelomično izlazne dretve. Mehanizam sinkronizacija dretve potrebno je predvidjeti jer se ne može očekivati da će se faze višedretvenog zadatka izvršavati jednako. Također nije problem samo sinkronizacija

dretve već i mogućnost da pojedina dretva obavlja neke aktivnosti jednu po jednu. Taj problem rješen je na način onemogućavanja prekida procesora koji osigurava dretvu koja radi na taj način da ne može biti prekinuta do onog trenutka kada dretva „sama ne odluči“ prekinuti takav način rada. Postoji još i tzv. međusobno isključivanje koje je dretvi potrebno ugraditi u radno okruženje kako bi mogla provesti pouzdanu organizaciju višedretvenih zadataka.

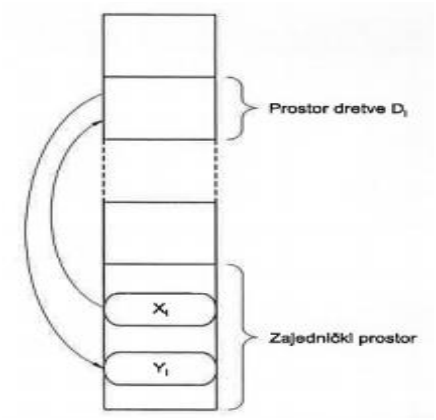
4.1.2. Model višedretvenosti

Tijekom izvođenja procesa, sam proces raspolaže sa svim dostupnim resursima koji su mu potrebni za izvršavanje zadataka. Također proces raspolaže i adresnim prostorom. Taj prostor je na raspolaganju svim dretvama. Svaka dretva ima na raspolaganju svoj spremnik koji je podijeljen na tri dijela. U prvom dijelu smještene su instrukcije dretve kao što je strojni program koji određuje izvođenje dretve. U sljedećem je smješten stog dretve, a u zadnjem dijelu smješteni su lokalni podaci dretve. U adresnom prostoru postoji još jedan dio koji je zajednički i kojeg mogu koristiti sve dretve.
Tamo se smještaju globalne ili zajedničke varijable od svih dretvi.



Slika 8. Podjela procesnog spremničkog prostora *(Izvor: Leonardo Jelenković: Operacijski sustavi- skripta 2015./2016.)*

Ukoliko kod izvršavanja zadataka postoji višedretvenost treba pretpostaviti da je u sustavu osigurana pravilna promjena konteksta te da se dretve mogu nesmetano odvijati, prekidati i nastavljati.



Slika 9. Dretva čita ulazne podatke iz svoje domene Xi i zapisuje svoje rezultate u svoju kodomenu Yi *(Izvor: Budin L., Golub M., Jakobović D., Jelenković L. : Operacijski sustavi, 3. Izdanje, Zagreb, 2013(str 65.))*

Da bi se bolje shvatilo ponašanje dretvi, pretpostavlja se da neka dretva Di u trenutku kada ona započinje svoj rad čita početne podatke iz jednog podskupa zajedničkog dijela spremnika (svoje domene Xi), na kraju izvođenja, dretva će ispisati svoje rezultate u jedan podskup zajedničkog spremnika(svoju kodomenu Yi). Također se može zamisliti da kada dretva započinje, ona prepisuje ulazne podatke iz Xi u svoje lokalne varijable. Nakon što obavi potrebnu obradu tako da koristi samo svoj dio adresnog prostora, dretva u trenutku kada završava svoj posao upisuje rezultate u svoju kodomenu. Po takvom načinu rada je slična potprogramu pomoću kojeg se prenose vrijednosti parametra.[7] Prema tome iz ovog primjera može se zaključiti da se izvođenjem dretve obavlja preslikavanje iz domene Xi u kodomenu Yi koje je prikazano na slici 9.

Postoji i druga vrsta zadataka za koje kažemo da su međusobno neovisni. Takvi zadaci nemaju zajedničku lokaciju u svojoj domeni i kodomeni niti su potrebni jedno drugome,a mogu se izvoditi i na različitim računalima. Iako se mogu izvoditi proizvoljnim redoslijedom, ako jedan podzadatak čita svoje ulazne podatke iz lokacije u koju drugi podzadatak upisuje svoje rezultate tada se podrazumijeva da će nakon izvođenja cijelog zadatka biti važan redoslijed izvođenja pripadajućih dretvi. Postoji i mogućnost da dva nezavisna podzadatka imaju zajedničku spremničku lokaciju. Ukoliko postoje lokacije iz koje podzadaci čitaju svoje ulazne podatke tada se njihov sadržaj neće mijenjati izvođenjem jednog ili drugog podzadatka. Oba podzadatka će pronaći jednake sadržaje na tim lokacijama bez obzira na redoslijed kojim su se izvodile njihove dretve.

[7] Budin L., Golub M., Jakobović D., Jelenković L. : Operacijski sustavi, 3. Izdanje, Zagreb, 2013. (str.65)

$$(X_i \cap Y_j) \cup (X_j \cap Y_i) \cup (Y_i \cap Y_j) = \emptyset$$

Slika 10. Uvjet nezavisnosti *(Izvor: Budin L., Golub M., Jakobović D., Jelenković L. : Operacijski sustavi, 3. Izdanje, Zagreb, 2013(str 66.))*

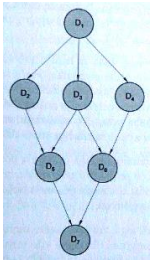
Uvjet nezavisnosti može se izraziti tako da presjek domene jednog i kodomene drugog te presjek njihovih kodomena budu prazni skupovi, ali se može dopustiti da presjek njihovih domena ne bude prazan što je prikazano slikom 10.

Kao što je u prethodnom poglavlju definirano složeni zadatak može se podijeliti na više podzadataka. Tijekom podjele na podzadatke potrebno je utvrditi zavisnost ili nezavisnost. Ako su zadaci zavisni tada je potrebno utvrditi njihov redoslijed izvođenja. Izvođenje podzadataka tijekom izvođenja pretvorit će se u sustav dretvi koje će zajedničkim djelovanjem obaviti zadani složeni zadatak. Kada su zadaci nezavisni tada se mogu izvršavati proizvoljnim redom samo ako nemaju zajedničku memorijsku lokaciju i ako im je dopušteno samo čitanje zajedničke memorijske lokacije.

4.1.3. Sustav dretvi- zavisnost i nezavisnost

Zadaci koji su zadani, najčešće se sastoje od podzadataka te se za svaki par zadataka utvrđuje nezavisnost ili zavisnost. Na taj način utvrđujemo redoslijed izvođenja podzadataka. Podzadaci pretvorit će se u sustav dretvi te će zajedno obaviti zadatak. Zavisnost ili nezavisnost pojedinih zadataka utvrđuje se tako da ako se pojedini zadaci mogu rastaviti na podzadatke tada se proučavanjem njihovih domena i kodomena utvđuje međusobna zavisnost ili nezavisnost koja se može prikazati usmjerenim grafom.

Slikom usmjerenog grafa se prikazuje primjena sustava dretvi (sustava podzadataka). Čvorovi grafa prikazuju pojedine dretve dok strelice predstavljaju redoslijed izvođenja pojedine dretve.



Slika 11. Primjena sustava dretvi *(Izvor: Budin L., Golub M., Jakobović D., Jelenković L. : Operacijski sustavi, 3. Izdanje, Zagreb, 2013(str 67.))*

D_1 početna dretva, a D_7 završna dretva. One dretve koje se nalaze na istom putu moraju se izvoditi određenim redoslijedom koji je propisan i te su dretve međusobno ovisne jedne o drugoj. Dretve koje se ne nalaze na istom putu mogu se izvoditi proizvoljnim redoslijedom, ali moraju biti međusobono neovisne. Za sve parove koji se nalaze na slici 11. treba utvrditi uvjet nezavisnosti to jest mora se utvrditi je li uvjet ispunjen. Sa primjera može se još vidjeti da nakon što završi dretva D_1, dretve D_2, D_3 i D_4 mogu se izvoditi proizvoljnim redoslijedom te na taj način se ubrzava izvođenje procesa. Ukoliko se na ovaj način rasčlanjuju zadaci na podzadatke koji su razmješteni na što više paralelnih puteva, takav način rasčlanjivanja je osnovni preduvijet za dobro iskorištavanje višeprocesorskih sustava. Dretve koje se nalaze na paralelnim putevima mogu se izvoditi paralelno na dva načina:

- istodobno- za njihovo izvođenje na raspolaganju je više fizičkih procesora
- prividno istodobno- izvode se na istom procesoru.

Ako imamo dretve istog procesa, one koje rješavaju jedan zajednički zadatak, tada se može očekivati mnogo veći broj zavisnih parova dretvi. Prilikom postavljanja programa treba obratiti pažnju na paralelizme u rješavanju nekog zadatka. Također treba pomno analizirati sa stajališta zavisnosti zadatka i izostaviti nepotrebne grane u grafu ako se utvrdi da su one nepotrebne u daljnjem rješavanju. Kako zavisne dretve obavljaju raznovrsne poslove, one moraju imati mogućnost razmjene podataka. Jedan od najjednostavnijih načina može se postići tako da jedna dretva piše u domenu drugu dretve. Dok je drugi način pomoću poruka koje moraju imati definirani oblik, a razmjenjuju se u posebno rezerviranom dijelu zajedničkog spremnika.

4.1.4. Međusobno isključivanje i cikličke dretve

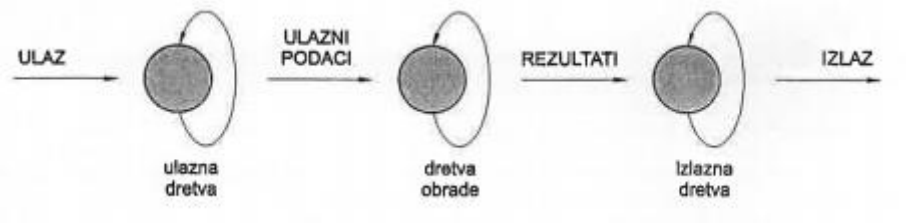
Mehanizam međusobnog isključivanja vrlo je bitan kod računalnog sustava koji omogućava da se neka sredstva koriste pojedinačno. Takav način rada dolazi do izražaja kod izvođenja dretvi na jednoprocesorskom sustavu, tada dretva osigura svoj rad tako da onemogućuje prekidanje. Kod onemogućenog prekidanja, nijedan prekid neće moći doći do procesora sve dok dretva ne obavi svoj kritični odsječak[8] te nakon toga programski omogući prekid i na taj način prelazi u nekritični odsječak. U slučaju višeprocesorskog sustava, zabrana prekida neće rješiti problem s obzirom na to da je mehanizam međusobnog isključivanja podloga za druge mehanizme.

[8] Kritični odsječak- dijelovi dretve koji koriste neko zajedničko sredstvo

Da bi se dogodilo međusobno isključivanje mora se zadovoljiti nekoliko uvjeta:

- samo jedna dretva može se nalaziti u kritično odsječku
- ako neka dretva zastane prije ulaska u kritični odsječak, to ne smije spriječiti ulazak druge dretve na ulazak u kritični odsječak
- izbor dretvi koja treba ući u kritični odsječak mora se obaviti u konačnom vremenu[9]

Mnoge operacije u računalnom sustavu se ponavljaju, najviše u operacijskom sustavu. Dretve koje se uzastopno ponavljaju nazivaju se cikličkim dretvama.



Slika 12. Cikličke dretve u ostvarenju cjevovodne obrade *(Izvor: Budin L., Golub M., Jakobović D., Jelenković L. : Operacijski sustavi, 3. Izdanje, Zagreb,2013(str69.))*

Slikom 12. Prikazana je ciklička dretva u ostvarenju cjevovodne obrade, trajno ponavljanje dretve označeno je granom koja uz čvor čini petlju.

Cjevovodno ponašanje računalnog sustava moglo bi se postići tako da se:

- Ulazna dretva uzastopce ponavlja prihvaćajući u svakom prolazu kroz petlju ponavljanja novu skupinu podataka i prenoseći ih dretvi obrade.
- Radna dretva prihvaća podatke koje joj predaje ulazna petlja, obrađuje ih te predaje rezultate izlaznoj dretvi i vraća se na početak gdje čeka nove rezultate
- Izlazna dretva prihvaća rezultate radne dretve, prosljeđuje ih u izlaznu napravu i vraća se na svoj početak gdje čeka nove rezultate.

5. Ostvarenje međusobnog isključivanja dviju dretvi

Kod odvijanja višedretvenih procesa na više procesorskim sustavima javlja se pitanje kako učinkovito ostvariti mehanizam međusobnog isključivanja koji osigurava da se određena sredstva koriste pojedinačno ako želimo postići da se dretve odvijaju istodobno uz uvijet da se jedna od njih nalazi u kritičnom odsječku, da mehanizam međusobnog isključivanja djeluje i kada su brzine izvođenja dretvi proizvoljne, kada se jedna od proizvoljnih dretvi u kritičnom

[9] Tanenbaum A. S., Bos H. - Modern operating systems, 4. izdanje, Amsterdam, 2015. (str.121), 30.07.2017.

odsječku mora izvršiti u konačnom vremenu i kada neka dretva zastane u nekritičnom dijelu, a ne smije sprečavati ulazak druge dretve u kritični odsječak. Da bi se pronašla dovoljno dobra rješenja za ove navede probleme, promatrat će se sustav sa dvije dretve. Ukoliko se dokaže da jedan od problema nije dobar za sustav sa dvije dretve, tada se to odnosi i na sustav sa više dretvi. Tek kada se dokaže da neki sustav funkcionira sa dvije dretve, tada se može razmatrati i za veći broj.[10]

5.1. Prvi pokušaj

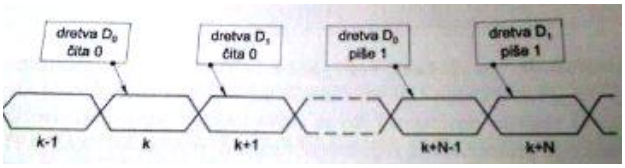
Kod prvog pokušaja međusobnog isključivanja uvodi se jedna varijabla proizvoljnog imena koja je smještena u zajedničkom procesnom prostoru koji je dostupan svim dretvama, u ovom slučaju dvjema. Neka se ova varijabla zove ZASTAVICA i neka ima dvije moguće vrijednosti: 0 ili 1. Dretva D_i se nalazi na svom kritičnom odsječku dok je druga D_j dretva ona koja želi ući u kritično odsjek, ona će u ovoj varijabli pronaći vrijednost 1 te će započeti sa izvođenjem petlje. Ta petlja će se izvoditi sve dok D_i ne zapiše u varijablu vrijednost 0. Tek nakon što se upiše vrijednost 0, dretva D_j će u varijablu moći upisati vrijednost 0 i ući u svoj kritični odsječak. Bitno je napomenuti da je kod dretve D_j koja se izvodi procesor j zauzet cijelo vrijeme te je u režimu radnog čekanja. Na taj način ne troši samo vrijeme procesora nego i sabirničke cikluse jer neprestano čita varijablu ZASTAVICA.

```
dok je (1) {
    čitati varijablu ZASTAVICA;
    dok je (ZASTAVICA != 0) {
        čitati varijablu ZASTAVICA;
    }
    ZASTAVICA = 1;
    kritični odsječak;
    ZASTAVICA = 0;
    nekritični osječak;
}
```

Slika 13. Prvi pokušaj *(Izvor: Budin L., Golub M., Jakobović D., Jelenković L. : Operacijski sustavi, 3. Izdanje, Zagreb, 2013(str 71.))*

Takav način pokušaja međusobnog isključivanja uopće ne daje zadovoljavajuće rezultate. Osnovni problem je u tome da dretva koja prva pročita vrijednost 0 nema dovoljno vremena da bi pohranila svoju vrijednost jer već sljedeći sabirnički ciklus dobiva svoj procesor..

[10] Budin L., Golub M., Jakobović D., Jelenković L. : Operacijski sustavi, 3. Izdanje, Zagreb, 2013. (str.70)



Slika 14. Događaj u kojem obje dretve istovremeno ulaze u kritični odsječak *(Izvor: Budin L.,Golub M., Jakobović D., Jelenković L. : Operacijski sustavi, 3. Izdanje, Zagreb, 2013(str 672.))*

Sljedeći problem javlja se kada obje dretve istovremeno žele ući u kritični odsječak te da obje pročitaju isto vrijednost sa varijable. Takav problem se također može javiti i kod jednoprocesorskog sustava ako se prvu dretvu koja je upravo pročitala varijablu, prekine prekidom nakon te instrukcije.

5.2. Drugi pokušaj

Kod drugog pokušaja također će se koristiti jedna varijabla. Ta varijabla imenom PRAVO poprimat će vrijednosti indeksa dretvi.

```
dok je (1) {
    čitati varijablu PRAVO;
    dok je (PRAVO != I) {
        čitati varijablu PRAVO;
    }
    kritični odsječak;
    PRAVO = J;
    nekritični osječak;
}
```

Slika 15. Primjer drugog pokušaja *(Izvor: Budin L., Golub M., Jakobović D., Jelenković L. : Operacijski sustavi, 3. Izdanje, Zagreb, 2013(str 73.))*

Ukoliko varijabla Pravo ima vrijednost ==I, tada dretva D_i smije ući u svoj kritični odsječak. I u ovom pokušaju nisu ispunjeni svi gore navedeni zahtjevi. Problem se javlja kada dretva D_i uđe u kritični odsječak, a varijabla PRAVO ima vrijednost I, tada D_i pohranjuje u varijablu vrijednost J. Ako dretva nakon toga stane u nekritični odsječak, D_j će moći jednom ući u svoj kritični odsječak i upisati vrijednost I u varijablu PRAVO. Dretva D_i stoji na svom nekritičnom odsječku i nikada neće iskoristiti svoje PRAVO niti će vratiti pravo ulaska dretvi D_j. Takvo rješenje moguće je jedino ako se dretve odvijaju naizmjenično u svojim odsječcima. Drugi pokušaj ukazuje na to da svaka dretva ima samo jednom pravo ući u kritični odsječak.

5.3. Treći pokušaj

Kod ovog pokušaja neće se koristiti jedna već dvije varijable. S obzirom da se u prva dva pokušaja koristila jedna varijabla u kojoj su se dretve i pisale i čitale. S takvim pristupom

dvije međusobno nezavisne dretve postale su zavisne.

```
dok je (1) {
     čitati varijablu ZASTAVICA[J];
     dok je (ZASTAVICA[J] != 0) {
          čitati varijablu ZASTAVICA[J];
     }
     ZASTAVICA[I] = 1;
     kritični odsječak;
     ZASTAVICA[I] = 0;
     nekritični odsječak;
}
```

Slika 16. Primjer trećeg pokušaja *(Izvor: Budin L., Golub M., Jakobović D., Jelenković L. : Operacijski sustavi, 3. Izdanje, Zagreb, 2013(str 74.))*

Uvode se dvije varijable koje će imati naziv ZASTAVICA[0] i ZASTAVICA [1]. To znači da dretva D_i mijenja vrijednost samo svoje ZASTAVICE, dok od suparničke dretve samo čita. Kada je vrijednost ZASTAVICA[I] ==0, dretva D_i se nalazi u nekritičnom odsječku, ako je ZASTAVICA[I]==1, dretva D_i se nalazi u kritičnom odsječku. Prema tome dretva koja želi ući u kritični odsječak, moći će ući samo kada se druga dretva nalazi u svom nekritičnom odsječku. Iako se problem koji se javljao u prethodna dva pokušaja pokušao rješiti uvođenjem dodatne varijable, niti to nije bilo dovoljno da bi se ispunili svi zahtjevi. Naime obje dretve mogu istovremeno pokušati ući u kritične odsječke na način da pročitaju vrijednost 0 u dva uzastopna sabirnička ciklusa i to sa zastavica suparničkih dretvi.

5.4. Četvrti pokušaj

U četvrtom pokušaju se dorađuje treći na način da se želi zaobići problem tako da se prije ispitivanja suprotne zastavice, prva dretva podigne svoju i na taj način označi da želi ući u kritični odsječak.

```
dok je (1) {
     ZASTAVICA[I] = 1;
     čitati varijablu ZASTAVICA[J];
     dok je (ZASTAVICA[J] != 0) {
          čitati varijablu ZASTAVICA[J];
     }
     kritični odsječak;
     ZASTAVICA[I] = 0;
     nekritični osječak;
}
```

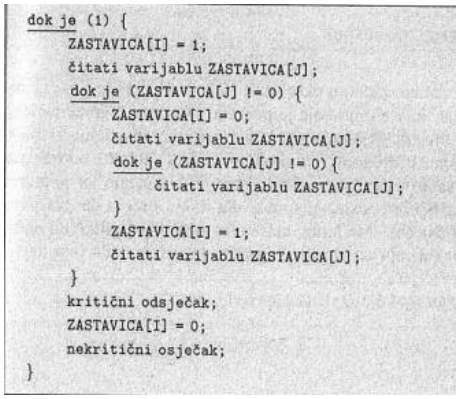
Slika 17. Primjer četvrtog pokušaja *(Izvor: Budin L., Golub M., Jakobović D., Jelenković L. : Operacijski sustavi, 3. Izdanje, Zagreb, 2013(str 74.))*

Problem se javlja kada obje dretve istovremeno žele ući u kritični odsječak. U tom slučaju obje dretve upisati će jedinicu u svoje zastavice te će nakon tog u ZASTAVICA[I] i ZASTAVICA[J] poprimiti vrijednosti 1. To za dretve znači da ulaze u petlje čekalice i tamo

ostaju zauvijek. Ovakva pojava naziva se još i potpuni zastoj ili dead lock.

5.5. Peti pokušaj

Svi pokušaji do sada bili su neuspješni u rješavanju svih navedenih zahtjeva.
Kako bi se izbjegao potpuni zastoj trebaju se modificirati dretve iz četvrtog pokušaja na način da dretva koja je podignula svoju zastavicu te nakon tog utvrdila da je i zastavica druge dretve podignuta, prva dretva bi svoju zastavicu morala nakratko spustiti i pričekati spuštanje druge zastavice.



```
dok je (1) {
    ZASTAVICA[I] = 1;
    čitati varijablu ZASTAVICA[J];
    dok je (ZASTAVICA[J] != 0) {
        ZASTAVICA[I] = 0;
        čitati varijablu ZASTAVICA[J];
        dok je (ZASTAVICA[J] != 0){
            čitati varijablu ZASTAVICA[J];
        }
        ZASTAVICA[I] = 1;
        čitati varijablu ZASTAVICA[J];
    }
    kritični odsječak;
    ZASTAVICA[I] = 0;
    nekritični osječak;
}
```

Slika 18. Primjer petog pokušaja *(Izvor: Budin L., Golub M., Jakobović D., Jelenković L. : Operacijski sustavi, 3. Izdanje, Zagreb, 2013(str 75.))*

Nakon pomne analize niti ovaj način nije korektno ostvarivanje kritičnog odsječka. I u ovom pokušaju dva procesora bi mogli svoje dretve izvoditi istovremeno ako bi :

- Obje dretve u dva sabirnička ciklusa podignule svoje zastavice
- Obje dretve pročitale u dva uzastopna ciklusa vrijednosti zastavice suprotstavljene dretve.
- Obje dretve ustanovile da je suprotstavljena zastavica podignuta i u dva sabirnička ciklusa spuste svoje zastavice
- Obje dretve u dva uzastopna ciklusa ustanovile da su zastavice spuštene, prođu kroz manju petlju čekalicu i podignu svoje zastavice
- Kod ispitivanja uvjeta veće petlje čekalice obje dretve ustanovile da su zastavice podignute i opet ih spuste.

U odnosu na prethodne neuspjele pokušaje, u ovom pokušaju više instrukcija se mora izvoditi istodobno te je vjerojatnost nastanka potpunog zastoja[11] manja.

[11] Potpuni zastoj (dead lock)- kada dretva ne može nastaviti sa radom zbog zahtjeva za nekim sredstvom koje je zauzeto od druge strane neke dretve.

5.6. Šesti pokušaj- Dekkerov postupak

Da ipak rješenje takvog zadatka ne bi izgledalo nemoguće zbog prethodno pet neuspjelih pokušaja, pobrinuo se nizozemski matematičar T. Dekker, a opisao ga je E. W. Dijkstra. Rješenje se temelji na kombinaciji prethodnih pokušaja i to drugog i petog. U drugom pokušaju korištena je varijabla PRAVO koja je dopuštala dretvama da se mogu izvoditi samo naizmjenično.

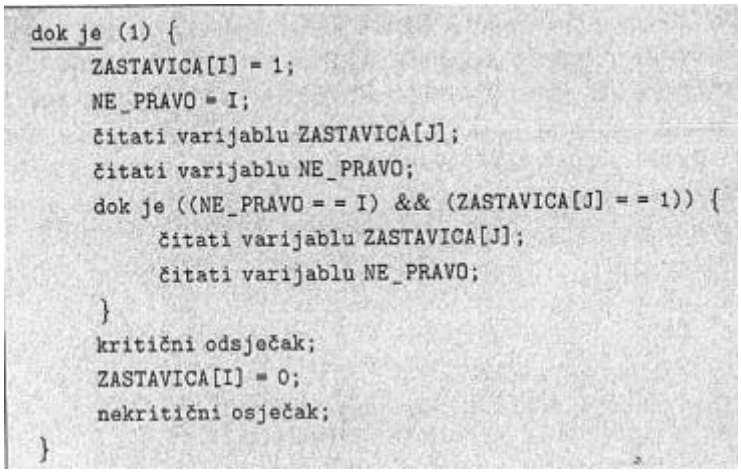
```
dok je (1) {
    ZASTAVICA[I] = 1;
    čitati varijablu ZASTAVICA[J];
    dok je (ZASTAVICA[J] != 0) {
        čitati varijablu PRAVO;
        ako je (PRAVO != I) {
            ZASTAVICA[I] = 0;
            dok je (PRAVO != I){
                čitati varijablu PRAVO;
            }
            ZASTAVICA[I] = 1;
        }
        čitati varijablu ZASTAVICA[J]
    }
    kritični odsječak;
    PRAVO = J;
    ZASTAVICA[I] = 0;
    nekritični osječak;
}
```

Slika 19. Primjer Dekkerovog pokušaja *(Izvor: Budin L., Golub M., Jakobović D., Jelenković L. : Operacijski sustavi, 3. Izdanje, Zagreb, 2013(str 76-77.))*

U ovom slučaju varijablu PRAVO koristit će se samo onda kada obje dretve konkuriraju za ulazak u kritični odsječak, tada će varijabla biti u mogućnosti odrediti redoslijed ulaska te na taj način pomoći u rješavanju problema međusobnog isključivanja. Ovakav pokušaj ne dopušta ulazak dretvama istovremeno u kritične odsječke te se na taj način ne može dogoditi potpuni zastoj.

5.7. Petersov postupak

Ovaj postupak sličan je Dekkerovom algoritmu. Varijablu poznatu od prije imenom PRAVO, preimenovat će se u NE-PRAVO. Svaka dretva koja želi ući u kritični odsječak postavlja svoju zastavicu i zapisuje svoj indeks u varijablu NE-PRAVO.



```
dok je (1) {
     ZASTAVICA[I] = 1;
     NE_PRAVO = I;
     čitati varijablu ZASTAVICA[J];
     čitati varijablu NE_PRAVO;
     dok je ((NE_PRAVO = = I) && (ZASTAVICA[J] = = 1)) {
          čitati varijablu ZASTAVICA[J];
          čitati varijablu NE_PRAVO;
     }
     kritični odsječak;
     ZASTAVICA[I] = 0;
     nekritični osječak;
}
```

Slika 20. Primjer Petersovog pokušaja *(Izvor: Budin L., Golub M., Jakobović D., Jelenković L. : Operacijski sustavi, 3. Izdanje, Zagreb, 2013(str 78.))*

Ako obje dretve žele istovremeno ući u kritični odsječak, tada varijabla NE-PRAVO poprima vrijednost indeksa one dretve čiji je procesor poslijedni dobio pravo pristupa i čeka tako dugo u petlji dok druga dretva ne spusti svoju zastavicu.Vrijednosti koje su bile upisane bit će izgubljene. Ukoliko dretve istovremeno postave zastavice, u petlji čekalici će ostati dretva koja je zadnja uspjela upisati svoj indeks. Nakon toga će druga dretva naići da uvjet za ponavljanje petlje nije ispunjen i tako ući u svoj kritični odsječak.

6. Lamportov algoritam

Prethodni brojni pokušaji rješavanja ovog problema bili su nedovoljno uspješni da bi zadovoljili sve zahtjeve u potpunosti. Do sad se moglo zaključiti da su potrebne najmanje dvije varijable da bi svi zahtjevi u potpunosti bili rješeni. Do prvog potpunog rješenja došao je L. Lamport. On je to rješenje nazvao pekarskim algoritmom[12]. Njegova zamisao bila je na primjeru prodavaonice u kojoj se dodijeljuju brojevi svakom kupcu. Na taj način spriječilo se guranje kupaca oko pulta te je svaki kupac u svakom vremenu znao na kojem se mjestu nalazi i koji se kupac poslužuje.

Ako se takav način upotrijebi na ove dretve to će značiti da će svaka dretva dobiti svoju varijablu u kojoj će biti zapisan broj koji je dretva dobila prilikom ulaska u kritični odsječak. Osim glavne varijable, svaka dretva dobiva još jednu tzv. nadzornu varijablu koja će označavati da se dretva upravo nalazi na početnoj fazi dodjele broja i da bi trebala pričekati sa ispitivanjem broja kojeg je dobila. Ni ovaj algoritam nije savršen pa tako i u ovom slučaju

[12] Budin L. i ostali, Operacijski sustavi, Zagreb, 2013 (str.80)

postoji mogućnost da dvije ili više dretvi žele istovremeno ući u kritični odsječak. Sve one mogu pročitati istu vrijednost zadnjeg dodijeljenog broja prije nego što se taj broj uveća. Da bi se to izbjeglo, treba sagledati Dekkerov pokušaj koji je to zamislio na način da sve dretve koje su željele ući u kritični odsječak su postavile svoju zastavicu i odabrala se ona dretva koja je pokazivala na varijablu PRAVO. Ta varijabla je mijenjala svoju vrijednost. Ako taj način primjenimo na Lamportov algoritam, vrijednost indeksa određivala bi pravo ulaska dretve ukoliko dođe do slučaja kada dvije dobiju iste brojeve. Pomoću početnog dodijeljenog indeksa određen je unaprijed redoslijed ulaska u kritični odsječak. Dokazivanje ovog algoritma može se i dalje provoditi, ali je dovoljno da nije moguće stvoriti scenarij odvijanja dretvi koji bi izazvao nepoželjne pojave.

```
uđi_u_KO (I)
{
    //uzimanje broja
    ULAZ[I] = 1;
    ZADNJI++;
    BROJ[I] = ZADNJI;
    ULAZ[I] = 0;

    //provjera i čekanje na dretve s manjim brojem
    za J=1 do N
    {
        dok je ( ULAZ[J] == 1 )
            ; //čeka se da dretva J dobije broj, ako je u postupku dobivanja

        dok je ( BROJ[J] != 0 &&
                ( BROJ[J] < BROJ[I] || ( BROJ[J] == BROJ[I] && J < I ) ) )
            ; //čekaj ako J ima prednost
    }
}

izađi_iz_KO (I)
{
    BROJ[I] = 0;
}
```

Slika 21. Program dretve

(Izvor:Leonardo Jelenković:Operacijski sustavi- skripta 2015./2016.)

Prema slici 21. nakon ulaska dretve, dodijeljuje se broj i ona započinje pregledavanje preostalih dretvi i traži zadnji dodijeljeni broj. Također prolazi i kroz svoje varijable, ali se tamo ne može zadržati. Dretva može zastati na dva mjesta:

- Dretva D_i zastaje ispitujući ULAZ [J] samo ako dretva traži dodjelu broja
- Dretva D_i zastaje dok dretva D_j koja ima BROJ[J] manji od BROJ[I] ne izađe iz kritičnog odsječka i zapise BROJ [J]=0.

Nakon prolaska prve petlje dolazi do druge gdje ostaje sve dok dretva čiji je broj manji od BROJ [I] ne vrati varijablu BROJ [J] vrijednost nula. Ako je neka dretva D_k dobila u međuvremenu svoj BROJ[K], a taj broj je sigurno veći od BROJ[J] tada ne smeta što ga više ne gledamo. Dok D_i čeka u petlji čekalici da joj D_j svoj BROJ[J] stavi na nulu, druga dretva koja nije pregledana moje dobiti svoj BROJ[K]. kod pregledavanja varijabli neće doći do zadržavanja D_i jer je taj broj sigurno veći od BROJ[I].

6.1. Sklopovska potpora međusobnom isključivanju

Sklopovska potpora izvedena je na drugačiji način nego mehanizmi međusobnog isključivanja. U ovom potpoglavlju definirat će se i drugi način međusobnog isključivanja. Kod prvog pokušaj rješavanja problema, problem u onom rješavanju bio je provjera zastavice i njezino postavljanje koje se moglo prekinuti nekom drugom dretvom, a koja je također mogla provjeriti vrijednost zastavice. Da bi se taj problem učinkovito rješio, uvodi se sklopovska potpora i to na način da se koriste dva uzastopna sabirnička ciklusa.

Slika 22. Korištenje dva uzastopna sabirnička ciklusa *(Izvor:Leonardo Jelenković:Operacijski sustavi- skripta 2015./2016.)*

U prvom ciklusu obavlja se čitanje sadržaja iz adresirane lokacije, a u drugu se upisuje u istu lokaciju[13]. Tijekom izvođenja instrukcija obavljaju se sljedeće aktivnosti:

- U prvom ciklusu se dobavi sadržaj adresirane lokacije i smješta se u jedan od registra procesora, u drugom ciklusu pohranjuje se u lokaciju vrijednost 1[14].
- U prvom ciklusu dobavi se sadržaj adresirane lokacije i smješta se u jedan registar procesora, a u drugom ciklusu se pohranjuje u tu lokaciju vrijednost koja ja bila prethodno pohranjena na tom mjestu[15].
- U prvom ciklusu dobavi se sadržaj adresirane lokacije i smješta se u jedan registar procesora, a u drugom ciklusu se pohranjuje u tu lokaciju vrijednost uvećana za jedan[16].
- U prvom sabirničkom ciklusu sadržaj adresne lokacije se dobavi i smjeste ga u jedan od registar procesora, dok se u drugom ciklusu upisuje u zadanu memorijsku lokaciju nova vrijednost ako je sadržaj registra jednaka stara vrijednost[17].

Za pobliže objasniti koristiti će se prva aktivnost na prvom pokušaju. Prva aktivnost još se naziva TAS[18] i ona omogućuje međusobno isključivanje prvog problema. Svaka od dretvi

[13] Procesor dobije pristup sabirnici može je koristiti trojako: za čitanje, pisanje, za nedjeljivo čitanje i pisanje (read-modify-cycle)
[14] Takve instrukcije imaju naziv ispitati i postaviti (test and set)
[15] Instrukcije imaju naziv zamijeniti(swap)
[16] Fetch and add
[17] Takve instrukcije nazivaju se usporedi i zamijeni (compare and swap)
[18] TAS- test and set

može konkurirati za ulazak u kritični odsječak nadziran varijablom ZASTAVICA te je prikazana slikom 13.

Ako je početna vrijednost varijable 0, tada prva dretva koja uspije izvesti instrukcije ispitati i postaviti, dobaviti vrijednost 0, a prije nego druga dretva uspije pročitati tu istu varijablu u nju pohraniti vrijednost 1. Sve dretve koje će pokušati ući u kritični odsječak ostat će u svojim petljama čekalicama. Prva dretva koja dohvati vrijednost 0 kada se ta vrijednost pojavi u varijabli ZASTAVICA, ući će u svoj kritični odsječak.

Kod ovakvog rješenja postoje dva nedostatka:

- Unaprijed se ne može odrediti redoslijed ulaska dretve u kritični odsječak
- Dretve koje žele ući u kritični odsječak izvode radno čekanje[19].

Da bi se radno čekanje svelo na minimum potrebno je organizirati računalni sustav na taj način da se uklone sve dretve koje čekaju na ulazak u kritični odsječak i da se procesor prepusti nekoj drugoj dretvi. Dretva se pokreće tek kada će moći ući u svoj kritični odsječak.

7. Potpuni zastoj

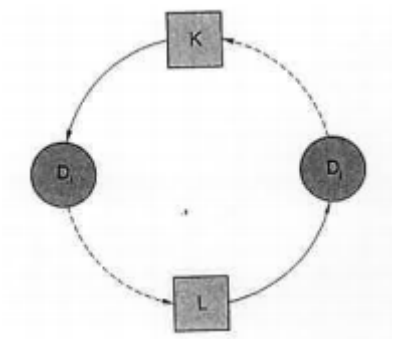
Potpuni zastoj ili dead lock javlja se u višeprogramskim sredinama. Više procesa može istovremeno zahtijevati upotrebu resursa računalnog sustava. To je proces koji čeka i ne mijenja svoje stanje zbog toga što je potrebno sredstvo zadržano od strane drugog procesa.

Uvjeti koji su potrebni za nastanak takvog zastoja su[20]:

- Moraju postojati barem dvije dretve i barem dva sredstva koje obje dretve koriste
- Sredstvo smije koristiti samo jedna dretva u isto vrijeme
- Sredstvo ne može biti oduzeto sve dok dretva sama ne otpusti
- Dretva drži dodijeljenjo sredstvo dok traži novo sredstvo.

Pomoću kritičnih odsječaka koji su zaštićeni binarnim semaforima može se omogućiti pojedinačna upotreba potrebnih sredstava.

[19] Kod radnog čekanja bespotrebno se troši vrijeme procesora i sabirničkih ciklusa
[20] http://www.zemris.fer.hr/~leonardo/os/fer/OS-skripta.pdf , 28.06.2017.



Slika 23. Petja potpunog zastoja
(*Izvor: Jelenković: Operacijski sustavi – skripta 2015../2016.*)

Krugovi predstavljaju dretve dok kvadrati predstavljaju sredstva koje dretve traže.

Crtkana strelica koja pokazuje na kvadrat označava da dretva zahtijeva sredstva dok strelica koja pokazuje na krug predstavlja sredstva koja dretva posjeduje. Također obje dretve sa slike nalaze se u petlji potpunog zastoja. U ovakvim jednostavnim primjerima potpuni zastoj je lako izbjeći tako da se zamijeni redoslijed poziva funkcija za ispitivanje binarnih semafora[21]. Ukoliko imamo više od dvije dretve i više od dva sredstva, pravilo binarnog semafora više ne vrijedi. za savladavanje potpunog zastoja kreirana su tri nužna uvjeta :

- Sredstvo u istom trenutku može upotrebljavati samo jedna dretva
- Dretvi sredstvo ne može biti oduzeto
- Dretva drži dodijeljeno sredstvo dok čeka na dodjelu dodatnog sredstva.

Da bi se spriječio zastoj potrebno je otkloniti prethodne uvjete. Najjednostavnije za otkloniti je ujedno i zadnji uvjet. Dretva bi morala primiti sva sredstva u isto vrijeme te da otpusti sva sredstva u isto vrijeme koje je do tada držala. Nakog toga opet bi tražila sva dostupna sredstva koja su joj potrebna za daljnji rad.

8. Jezgra operativnih sustava

S obzirom na poteškoće koje se događaju tijekom neposrednog međudjelovanja dretvi koje se mogu vidjeti u prethodnim primjerima, logično je potaknuti dretve da međusobno surađuju ili ih potaknuti da se natječu koja će preuzeti neka sredstva na organiziran način. Takvi zahtjevi za dretve mogu se realizirati samo na način da se pozivaju odgovarajuće funkcije jezgre. Jezgra operacijskog sustava je centralni dio OS-a. Glavna zadaća je upravljanje sklopovljem na najnižem nivou. Ona upravlja resursima i prestavlja sloj između korisničkih programa i

[21] Binarni semafor služi za sinkronizaciju dretvi i može imati dva stanja: prolazno i neprolazno

fizičkog sklopovlja.[22] Kod jezgre postoji više elemenata, a to su:

- upravitelj datotekama,
- elementi za komunikaciju sa uređajem,
- upravitelj memorijom,
- organizator
- dodjeljivač resursa[23].

Upravitelj datotekama (file manager) nadgledava korištenje uređaja za masovnu pohranu. To znači da prati gdje je spremljena datoteka, tko sve ima ovlasti nad njom i vodi računa o dostupnom mjestu za nove ili nadopunu već postojećih datoteka. Također dozvoljava i grupiranje datoteka i dokumenata u skupine. Takvim načinom spremanja, korisnici imaju organizirane vlastite podatke na njima prihvatljiv način. Slijedeći u nizu element je za komunikaciju sa uređajima. On predstavlja dio jezgre koji je zadužen za komunikaciju sa perifernim uređajom da bi prenio ono što korisnik želi učiniti sa tim uređajem. Svaki D.D. je kreiran za svoj tip uređaja i prevodi naredbe u konkretne korake koji osiguravaju izvršenje akcije. Upravitelj memorijom u slučaju da se na računalu izvodi samo jedan program ima minimalne dužnosti u kojima je odgovoran za kodiranje kod zauzimanje memorije. Problemi nastaju kada je u višezadatkovnom načinu rada, tada računalo mora istovremeno izvršiti više zahtjeva. U takvim slučajevima mora biti više programa u samoj radnoj memoriji i samim time više blokova podataka koje programi trebaju za svoj rad, a o svim zahtjevima računa vodi upravitelj memorijom.

Organizator je zadužen za ažuriranje podataka o procesima koji su aktivni, uvodi nove procese, briše trag od završenih procesa. Da bi uspješno odradio svoje zadatke mora stalno ažurirati informacije u memoriji koje se nazivaju „tablicom resursa“. U tablicu se upisuje novi podatak prilikom zaduživanja nove aktivnosti. Tom podatku pridružene su informacije kao što su prioritet izvođenja procesa, da li je proces u izvođenju ili čekanju te koliko je memorije dodijeljeno tom procesu.

Element dodjeljivač vodi računa o tome da li se proces koji je upisan u tablicu stvarno i izvodi. Kod sustava sa raspodijeljenim vremenom, provjera se vrši dijeljenjem vremena na segmente koji se nazivaju još i vremenski kvanti(odsjeci) i pridavanjem pažnje procesora određenom procesu tijekom dodijeljenog vremena u kojem je procesu dopušteno se izvoditi.

Također se i sastoji od skupine funkcija koje se mogu pozivati sklopovskim ili programskim prekidima. Jezgra funkcionira na način da kad dretva želi komunicirati sa drugom dretvom,

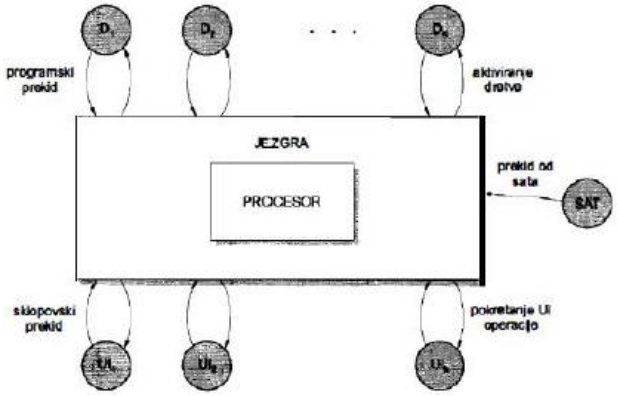
[22] https://sysportal.carnet.hr/node/75 , 06.06.2017.
[23] http://tvolaric.com/preuzimanja/operacijski_sustavi.pdf , 15.06.2017.

tada se pozivaju odgovarajuće funkcije jezgre koje obavljaju njihov zadani posao. Jezgra se ne poziva samo kada treba izvršiti međusobnu komunikaciju između dretvi nego i kada dretva želi obaviti ulaznu ili izlaznu operaciju. Nakon obavljenog posla od strane dretve, javlja se jezgri te ona zaustavlja njezino daljnje izvođenje. Jednostavan model jezgre temelji se na pretpostavkama da :

- su u adresnom prostoru procesa smješteni svi dretveni prostori
- čitav adresni prostor procesa je dohvatljiv dretvama
- se izvođenje dretvi obavlja u jednoprocesorskom sustavu
- da dretve mogu pozivati jezgrine funkcije odgovarajućim programskim prekidom
- u sustavu postoje i ulazne i izlazne naprave koje se uzimaju u razmatranje kako bi se jezgrom obuhvatila obrada sklopovskih prekida[24].

U takvom sustavu djeluje sistemski sat[25] koji funkcionira tako da se namjesti da broji impulse generatora takta koji trajno pohranjuje sve podatke o vremenu sa velikom preciznošću. Iz sata se mogu dobiti i periodički impulsi i na taj način se mogu izazivati periodni prekidi . Takvi prekidi služe za generiranje vremenskih intervala i za prekidanje izvođenja dretvi koje traju predugo. U sustavu se mogu dogoditi tri vrste prekida:

- sklopovski prekidi ulazno-izlaznih naprava
- periodni sklopovski prekidi sata
- programski prekidi izazvani dretvama.



Slika 24. Prikaz jezgre *(Izvor: Budin L., Golub M., Jakobović D., Jelenković L. : Operacijski sustavi, 3. Izdanje, Zagreb, 2013. (str 91))*

Grafičkim prikazom se može lakše predočiti kako izgleda jezgra i dretve te u kojem se dijelu aktivnosti pokreće sat koji pomoću prekida poziva funkcije jezgre.

Može se zaključiti da se u procesu poziva funkije jezgre i ulaska u jezgru događa pomoću

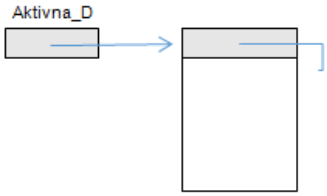
[24] Budin L., Golub M., Jakobović D., Jelenković L. : Operacijski sustavi, 3. Izdanje, Zagreb, 2013. (str.90)
[25] Sistemskim satom opremljeno je svako računala te kada se taj sat uskladi sa astronomskim , služe za očitavanje vremena.

prekida dok se izlazak iz jezgre događa pokretanjem dretve.

Sve dretve svoje važne podatke u jezgri moraju pohraniti. Svi važni podaci dretve smještaju se u jedan zapis kojeg se naziva opisnik ili deskriptor dretve. Tamo se još smještaju i kazaljke koje omogućuju lakše premještanje iz jedne u drugu dimamičku strukturu.

8.1. Aktivno stanje dretve

Kod jednoprocesorskih sustava samo jedna jedina dretva može biti aktivna, a to je dretva čije se instrukcije u tom trenutku izvode.

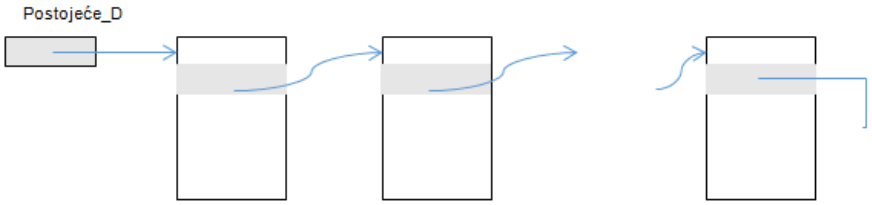


Slika 25. Lista postojeće dretve
(Izvor: Budin L., Golub M., Jakobović D., Jelenković L. : Operacijski sustavi, 3. Izdanje, Zagreb, 2013. (str 93))

U listi aktivne dretve mora se nalaziti samo jedan opisnik dretve. Tijekom pohranjivanja, sadržaj registra procesora smješta se na već unaprijed određeno mjesto u opisniku u listi. Kada se aktivna dretva izvrši, opisnik se premješta na drugo mjesto, a u listu dolazi nova dretva koja je u tom trenutku aktivna tako da se iz reda „Pripravna_D“ premjesti u red „Aktivna_D.

8.2. Pasivno stanje dretve

U listu se smještaju sve dretve koje se nalaze u adresnom prostoru nekog procesa. Na taj način, ukoliko ima potrebe za pristup opisnicima, oni se mogu pregledavati redom.



Slika 26. Lista postojeće dretve *(Izvor: Budin L., Golub M., Jakobović D., Jelenković L. : Operacijski sustavi, 3. Izdanje, Zagreb, 2013. (str 93))*

Prva kazaljka po redu u svim zapisnicima služi za povezivanje drugih lista te se na taj način može obići sve opisnike u listama bez obzira gdje se nalazili. Povezivanje se obavlja na drugom po redu kazaljkom. Pasivno stanje postojeće dretve događa se u trenutku kada se opisnik nalazi samo u listi „Postojeće_D“.

Na taj način uviđamo da se ne nalazi u niti jednoj drugoj listi te se na taj način nalazi u pasivnom stanju.

8.3. Pripravno stanje dretve

Kao i kod aktivnog stanja dretve i u ovom slučaju je moguće samo da se izvodi jedna dretva u tom trenutku, sve ostale su u procesu čekanja na dodjelu procesora. Kod takvih stanja dretvi javlja se pitanje koja dretva ima prednost nad drugom da postane aktivna. Jedno od rješenja je da se dodjela procesora dretvi događa po onom redoslijedu kojem je postala pripravna za izvođenje. Na taj način dretva koja je pripravna za izvođenje dolazi na kraj reda dok dretva čiji je opisnik na početku liste bit će prva u izvođenju. Da bi se izbjegla ovisnost trajanja umetanja o duljini reda, umetnute su dvije kazaljke. Prva pokazuje početak reda, a druga kraj reda. Onaj koji je prvi pristupio prvi će sa sigurnošću i biti poslužen (FCFS[26]).
Sljedeće rješenje može se temeljiti na prioritetima kod dretvi. Ako se upiše prioritet u dretvu tada bi se opisnike trebalo svrsati u listu prema prioritetima. Ako se želi postići takav način tada tada tijekom umetanja treba pregledati cijeli red da bi se ustanovilo na koje mjesto mora doći nova dretva. Da bi se to izbjeglo i smanjilo vrijeme i složenost mogu se uvesti redovi koji će sadržavati dretve iste prioritetnosti. Dretve koje se nalaze u redu sa najvećim prioritetom izvršavaju se prve, nakon toga one manje sve dok ne dođe do najnižeg prioriteta. Nakon izvršavanja svih redova prioriteta dretvi, procesor nema nikakvih zaduženja te je „slobodan“. Da procesor ne stoji kada nema dretvi za izvršavanje, tu dolazi latentna dretva. Takva dretva niti šteti procesoru niti ga koristi samo troši vrijeme tog procesora i čeka novi zadatak za njega. Ona se aktivira kada nema niti jedne dretve u redu pripravnosti. Ukoliko je sustav dvorazinski, u trenutku uvođenja latentne dretve, on postaje trorazinski.

8.4. Blokirane dretve

Kod izvođenja dretvi može se dogoditi da budu blokirane. Blokiranje dretve se događa ukoliko nije ispunjen neki uvjet za njihov rad. Dretve mogu biti blokirane na četiri načina :

- Čekanje na binarnom semaforu
- Čekanje na općem semaforu
- Čekanje isteka zadanog intervala kašnjenja
- Čekanje završetka ulazno-izlazne operacije

[26] First-come, first-served

Binarni semafor definiran je kod međusobnog isključivanja dretvi u poglavlju 6.
Razlika između binarnog i općeg semafora je ta što opći semafor može poprimiti i druge vrijednosti cijelog broja , za razliku od binarnog koji može samo 0 ili 1. Opći funkcionira na način da kada dretva prolazi kroz semafor, uključuje se jezgra koja smanji dretvninu vrijednost za jedan i ako je nakon toga vrijednost >= od 0, dopušta joj prolaz. Ako je dretva <= od 0 tada je blokirana i kada neka dretva zahtjeva postavljanje općeg semafora to će potaknuti povećanje vrijednosti te se na taj način ispunjava uvjet za pokretanje blokirane dretve. Odgađanje izvođenja zadanog intervala kašnjenja također može uzrokovati blokiranje dretve. Interval kašnjenja može biti cjelobrojni višekratnik kod perioda otkucaja sata. Nakon svakog prekida sata, vrijednost koja je pohranjena u opisnik pada za jedan. Kada padne na nulu, dretva koja se nalazi u redu odgođene dretve prebacuje se u red pripravnih dretvi. Ukoliko se više dretvi izvode odgođeno, one se u red odgođenih dretvi smještaju sortirane. Zadnje blokiranje dretvi događa se zbog čekanja ulazno-izlaznih operacija. Takve operacije dretve mogu obavljati samo preko jezgri. Navedene operacije dretva može koristiti samo pojedinačno. Svakoj toj operaciji mora se dodijeliti binarni semafor koji dretva mora proći. Kad dretva prođe binarni semafor, poziva se funkcija jezgre s kojom se obavljaju operacije. Sve dretve iz sustava mogu obavljati svoje operacije i ako je takav slučaj može se dogoditi da su one u nekom trenutku sve blokirane. Procesor ne radi ništa i čeka da se dogodi neki prekid koji će aktivirati barem jednu dretvu.

9. Ostvarivanje jezgre u višeprocesorskom sustavu

Do sada se pretpostavljalo da se sve dretve odvijaju u jednoprocesorskom sustavu te da se jezgrene funkcije odvijaju međusobno isključeno. U višeprocesorskom sustavu zabrana prekidanja u jednom procesoru ne osigurava međusobno isključivanje. U takvom sustavu svi procesori mogu adresirati vlastiti spremnik i jedan zajednički spremnik. Kod promatranja spremnika dretvi koje se izvode unutar jednog procesa do sada se pretpostavljalo da sve dretve mogu dohvaćati cijeli prostor za spremanje. Kako se sve manje koristi jednoprocesorski, a sve više višeprocesorski sustavi mogu se pretpostavljati različiti načini upotrebe spremničkog prostora. Zbog toga se može zanemariti postojanje lokalnih spremnika i pretpostavljati da se svi dretveni prostori nalaze u zajedničkom spremniku. Također se pretpostavlja da su instrukcije dretvi smještene u lokalnim spremnicima dok se lokalni

podaci i zajednički prostor nalazi u zajedničkom spremniku. Za kraj, pretpostavlja se da se cijeli dretveni prostor nalazi po pojedinim lokalnim spremnicima, a da se u dijeljenom spremniku nalazi i zajednički prostor svih dretvi. Kako su mogućnosti podjele poslova raznovrsne, posao u višeprocesorskom sustavu može se organizirati na način da se dretve izvode homogeno ili nehomogeno. Homogeno predstavlja sustav u kojem se sve dretve mogu izvoditi u bilo kojem procesoru što znači da se izvođenje dretvi može prepustiti bilo kojem procesoru koje pak olakšava njihovo raspoređivanje, dok nehomogeno kada se dretve mogu izvoditi samo na određenom procesoru i mora postojati mogućnost dodjeljivanja dretvi pojedinom procesoru. Bez obzira na upotrebu procesora, struktura podataka jezgre mora se nalaziti na dijeljenom spremniku. Također ona mora biti dohvatljiva svim dretvama bez obzira na kojim se procesoru izvode. Kod oba sustava uobičajeno je da se pripravne dretve organiziraju po procesorima zbog učinkovitijeg korištenja priručnog spremnika. Takvo organiziranje sprječava da se dijelovi dretvi zadržavaju u priručnim spremnicima procesora između dva uzastopna aktiviranja jedno te iste dretve. Jezgra operacijskog sustava mora voditi računa o raspodjeli vremena između dretvi te će ih ponekad premještati iz jednog procesora na drugi.

Zaključak

U ovom završnom radu govori se o upravljanje procesima ali najviše se tražilo rješenje za međusobno isključivanje kod izvođenja dretvi te pronalazak najpovoljnijeg rješenja. Funkcije operativnog sustava kod svih sustava imaju jednake zadaće koje korisniku olakšavaju jednostavnu upotrebu, višeprogramski rad i učinkovitost što je ujedno i cilj OS-a. Najviše pažnje u ovom radu posvećeno je procesu kojeg je najjednostavnije definirati kao program koji je zapisan ili pohranjen u strojnom obliku te se sastoji od niza instrukcija koje procesor mora razumijeti i ispravno obaviti prilikom izvođenja. Tijekom izvođenja tih instrukcija, proces nailazi na mnoge probleme koji se danas pokušavaju što jednostavnije i učinkovitije rješiti, a oni najčešći problemi opisani su u ovom radu.

OS koji se razvijaju danas, korisnicima omogućuje upravljanje računalom s lakoćom i obavljanje sve većeg broja zadataka istovremeno. Zahtjevi krajnih korisnika često su jednostavnost korištenja, pouzdanost i stabilnost sustava.

S obzirom na napredovanje na dnevnoj bazi kod OS, nitko ne može jasno predvidjeti kako će u budućnosti takvi sustavi funkcionirati i da li će uopće postojati u ovakvom obliku kao što ih poznajemo danas i sa problemima s kojima su danas programeri suočeni. Svaka promjena koja se dogodi kod OS je želja da se u potpunosti prilagodi potrebama krajnjih korisnika. Ukoliko će i ubuduće vodeće tvrtke koje posjeduju danas najzastupljenije OS-e na svijetu slušati želje i zahtjeve korisnika te se prilagoditi tržištu i tehnologijama budućnosti i dalje će poslovati sa ovakvim uspjehom kojeg imaju danas.

Literatura

a) Knjige:

1. Budin L., Golub M., Jakobović D., Jelenković L. : Operacijski sustavi, 3. Izdanje, Zagreb, 2013.

2. Tanenbaum A. S., Bos H. - Modern operating systems, 4. izdanje, Amsterdam, 2015.

b) Web:

1. http://www.istratech.hr/wp-content/uploads/2012/11/case2012_1.pdf, 10.06.2017.
2. http://www.zemris.fer.hr/~leonardo/os/fer/OS-skripta.pdf , 28.06.2017.
3. http://tvolaric.com/preuzimanja/operacijski_sustavi.pdf, 15.06.2017
4. https://sysportal.carnet.hr/node/75, 06.06.2017.

Popis slika