**OPERACIJSKI SUSTAVI**

Literetura: Abraham Silberschatz, Peter B. Galvin, "Operating System Concepts", Addison-Wesley Publishing Company, 1994.

Sadržaj predmeta:

1. Uvod
   1. Definicija operacijskog sustava
   2. Kratak povijesni pregled razvoja operacijskih sustava
2. Arhitektura računarskog sustava
3. Struktura operacijskog sustava
4. Upravljanje procesima
   1. Definicija i svojstva procesa
   2. Dodjela procesora
   3. Sinkronizacija među procesima
   4. Potpuni zastoj
5. Upravljanje memorijom
   1. Postupci dodjele memorije
   2. Virtualna memorija
6. Upravljanje sustavom datoteka
   1. Sustav datoteka
   2. Postupci implementacije sustava datoteka
   3. Sekundarna memorija
7. Zaštita i osiguranje računarskog sustava
8. Distribuirani operacijski sustavi

# UVOD

*Operacijski sustav* je program koji djeluje kao sučelje između korisnika i sklopovlja računarskog sustava. Njegova namjena je da osigura okoliš u kojem korisnik može izvoditi programe. Primarni cilj operacijskog sustava je da učini računarski sustav *prikladan* za korištenje. Sljedeća njegova uloga je da osigura *učinkovitu* uporabu resursa računarskog sustava.

Da bi se shvatilo što je to operacijski sustav, potrebno je osvrnuti se na njegov povijesni razvoj. To je razlog da se prvo napravi povijesni pregled operacijskih sustava. Prolazeći kroz različite faze razvoja operacijskih sustava, uočava se kako su se isti razvijali rješavajući rastuće zahtjeve korisnika i povećane mogućnosti sklopovlja.

## Što je operacijski sustav

Operacijski sustav je važna sastavnica svakog računarskog sustava. Računarski sustav grubo se može podijeliti na slijedeće četiri sastavnice (slika 1.1.1.):

* *sklopovlje,*
* *operacijski sustav,*
* *aplikacijske programe,*
* *korisnike.*



**Slika 1.1.1:** Prikaz računarskog sustava.

Sklopovlje se sastoji od procesne jedinice (CPU *Central Processing Unit*), memorije, ulazno/izlaznih naprava (diskova, disketnih jedinica, zasuna, tipkovnica). Korisnički programi su programi prevodioci (*compilers*), sustavi za obradu baza podataka (*database systems*), programi za uredsko poslovanje (*office*), programi za obradu teksta ili tekst editori (Word), programi za crtanje (AutoCad, CorelDraw). Pomoću ovih programa korisnik koristi resurse računarskog sustava za rješavanje različitih poslova. Naravno postoje različiti korisnici (ljudi, strojevi, druga računala) s različitim znanjima i zahtjevima koji pokušavaju rješavati postavljene zadatke. S obzirom na navedeno postoje i različiti aplikacijski programi. Operacijski sustav upravlja i koordinira uporabu resursa računala od strane različitih aplikacija.

Operacijski sustav može se usporediti s *državom* odnosno *upravom*. Dijelovi računarskog sustava su: *sklopovlje, programska podrška i podaci*. Operacijski sustav omogućava ispravnu uporabu svih resursa tijekom rada računarskog sustava. Kao i uprava, on sam za sebe ne provodi nikakav djelatan rad. On jednostavno osigurava *okoliš* u kojem program može učinkovito riješit postavljeni zadatak.

*Dodjela resursa* je također zadatak operacijskog sustava. Da bi se riješio postavljeni problem potrebni su različiti resursi računarskog sustava, sklopovlje, programi i podaci. Operacijski sustav dodjeljuje potrebne resurse računarskog sustava svakom pojedinom zadatku. Kako različiti zadaci mogu postavljati zahtjeve za istom komponentom računarskog sustava, operacijski sustav odlučuje kojem zadatku će biti dodijeljen taj resurs. Učinkovitost računarskog sustava kao i pravičnost su kriteriji na osnovi kojih operacijski sustav mora donijeti ovu odluku.

Nešto različit pristup objašnjenju što je operacijski sustav zasnovan je potrebi upravljanja različitim ulazno/izlaznim napravama i korisničkim programima. Tako se operacijski sustav definira kao *upravljački program*, koji upravlja izvođenjem korisničkih programa u cilju otklanjanja pogrešaka kao i pogrešne uporabe sklopovlja. Ovo se posebno odnosi na uporabu i upravljanje ulazno/izlaznim napravama.

Općenito, ne postoji jednoznačna definicija operacijskog sustava. Kako bi se realizirao uporabljivi računarski sustav nastao je i operacijski sustav. Osnovna namjena računarskog sustava je da izvodi korisničke programe, odnosno da rješava zadatke postavljene od strane korisnika. Kako bi se ostvari ovaj cilj konstruirano je sklopovlje računala. Ali ono samo za sebe nije posebno prikladno za uporabu. U cilju pojednostavljenja uporabe sklopovlja razvijena je odgovarajuća programska podrška koja upravlja radom sklopovlja (*device drivers*). Ona je proširena programima koji odlučuju kada i na koji način je dozvoljeno pojedinom programu pristup određenim resursima računarskog sustava. Ovim su zadaci upravljanja i dodjele resursa računarskog sustava objedinjeni u jedinstvenu cjelinu, operacijski sustav.

Može se zaključiti da ne postoji jedinstvena definicija što je to operacijski sustav. Odavde proizlazi da se ne može ni odrediti koji programi su sastavni dio operacijskog sustava a koji ne pripadaju operacijskom sustavu. Najjednostavnije je zaključiti da su sistemski programi oni programi koje proizvođač isporučuje sa sklopovljem (računalom). Naravno, računala se međusobno znatno razlikuju s obzirom na sklopovlje, procesor i njegovu arhitekturu, memoriju, diskove, pisače te ostale ulazno/izlazne naprave. Tako se i isporučena programska podrška može znatno razlikovati, primjerice najjednostavnija verzija DOSa s jednostavnim *command interpreter-*om, pa sve do WINDOWS 95 ili WINDOWS NT, s grafičkim sučeljem, editorima i sl. Često se također definira da je operacijski sustav program koji je stalno aktivan u računarskom sustavu i naziva se *jezgra* (*kernel*), a sve ostalo su aplikacijski programi.

Uobičajenije je da se operacijski sustav definira prema onome što on radi, a ne prema onome što je on. Na osnovi dosada navedenoga može se zaključiti da je osnovni zadatak operacijskog napraviti računarski sustav što *prikladnijim* za uporabu. Slijedeći zadatak operacijskog sustava je da osigura što *efikasniju* uporabu računarskog sustava. Ovo je naročito važno kod velikih i skupih višekorisničkih sustava.

Konačno može se zaključiti da su *uporabljivost* i *efikasnost* računarskog sustava osnovni zadaci operacijskog sustava.

Sljedeći korak je pogledati kako su se operacijski sustavi razvijali kroz relativno kratku tridesetogodišnju povijest računarstva. Uočiti će se veza između sklopovlja (*hardware*) i operacijskih sustava. Da bi se pojednostavnila uporaba sklopovlja razvijali su se operacijski sustavi. Kako su isti razvijani i primjenjivani, uočavale su se potrebe za promjenama i novinama u sklopovlju kojim bi se pojednostavnio i učinio učinkovitiji operacijski sustav. Kratak povijesni pregled ukazuje kako je razvoj i zahtjevi na operacijski sustav rezultirao novinama u sklopovlju.

**2. Struktura operacijskog sustava**

Kako je već u uvodu napomenuto operacijski sustav osigurava okolinu za izvođenje korisničkih programa. Kao i u svakom postupku projektiranja tako i kod projektiranja operacijskog sustava važno je dobro odrediti ciljeve koje je potrebno ostvariti. Računarski sustav za koji se operacijsku sustav izrađuje predstavlja polazište projektiranju operacijskog sustava.

Postoji više važnih točaka koje je potrebno sagledati prilikom projektiranja operacijskog sustava. Prva od značajki je ispitivanje koje sve usluge operacijski sustav mora osiguravati. Sljedeća je odrediti veze sustava s korisnicima i programerima. Treća je razlaganje sustava na sastavnice i određivanje njihovih međuveza.

**2.1. Sastavnice operacijskog sustava**

Ovako složen programski paket kao što je operacijski sustav može se ostvariti samo razgradnjom problema na sastavnice i nezavisnim rješavanjem i ispitivanjem svake od sastavnica. Svaku sastavnicu potrebno je dobro odrediti, tj. odrediti o kojem se dijelu računarskog sustava radi, koje su mu funkcije i koji su mu ulazi i izlazi. Naravno svi računarski sustavi nemaju jednaku strukturu i zahtjeve, pa se i primijenjeni operacijski sustavi razlikuju po veličini i svojim funkcijama.

Na osnovi dosadašnjih razmatranja vezanih uz arhitekturu i strukturu računarskih sustava mogu se izdvojiti sljedeći zadaci operacijskog sustava:

1. upravljanje procesima,
2. upravljanje radnom memorijom,
3. upravljanje sekundarnom memorijom,
4. upravljanje ulazom/izlazom,
5. upravljanje datotekama,
6. zaštita dijelova sustava,
7. otkrivanje pogrešaka u radu sustava,
8. tumačenje upravljačkih naredbi,
9. upravljanje mrežom računala.

**Upravljanje procesima**

Pod pojmom proces u računarskom sustavu podrazumijeva se program koji se izvodi. Tako se proces definira kao program u izvođenju. Program sam za sebe je pasivan entitet, zapisan kao datoteka na disku dok je proces niz naredbi koje procesor izvodi aktivan entitet.

Za svoje izvođenje proces zahtjeva određene resurse računarskog sustava, procesor, memoriju, sekundarnu memoriju, datoteke, ulazno/izlazne jedinice. Ovi resursi dodjeljuju se procesu ili prije nego što je on započet ili tijekom izvođenja. Tako npr. jedan editor kao ulazne podatke ima izgled izbornika, ali kada se učitava neka datoteka na obradu, s tipkovnice ili mišem se odabire datoteka koju ovaj proces uzima u obradu. Datoteka s izbornikom predstavlja resurs koji se dodjeljuje procesu prije njegovog izvođenja, a datoteka s podacima je resurs koji ovaj proces zahtjeva tijekom izvođenja. Nakon što proces više ne koristi neki resurs on ga oslobađa. Taj resurs sada može stajati na raspolaganju nekom drugom procesu.

Program se izvodi slijedno naredbu po naredbu. Prema tome i proces se izvodi slijedno, pa u svakom vremenskom trenutku postoji naredba koja se izvodi za neki proces. Program koji se izvodi može se dijeliti u više procesa. Tako npr. program može zahtijevati upis na disk kao jedan proces i nastaviti s obradom nekih drugih nezavisnih podataka kao drugi proces. Tako je uobičajeno da program tijekom izvođenja generira veći broj različitih procesa.

Prema navedenom proces je radna jedinica u sustavu za obradu. U takvom se sustavu nalazi skup procesa, neki su sistemski procesi ili procesi operacijskog sustava, a drug korisnički procesi odnosno procesi koji su dio korisničkih programa. Svi ovi procesi mogu se izvoditi istovremeno vremenski dijeleći procesor.

Operacijski sustav mora vezano uz upravljanje procesima obavljati sljedeće:

* stvaranje i poništavanje korisničkih i sistemskih procesa,
* odgađanje odnosno prekidanje i ponovno aktiviranje procesa,
* sinkronizacija među procesima,
* komunikacija među procesima,
* razrješavanje potpunog zastoja (*deadlock*).

**Upravljanje radnom memorijom**

Radna memorija je veliko polje riječi koje iz kojeg procesor dohvaća naredbe i podatke preko adresa dodijeljene svakoj pojedinoj lokaciji. Kako bi se program izvodio moraju mu biti dodijeljene apsolutne adrese te mora biti unesen u radnu memoriju. Tijekom izvođenja programa, naredbe i podaci dohvaćaju se postavljajući memoriji njihove apsolutne adrese. Po završetku programa oslobađa se memorijsko područje zauzeto programom i podacima.

U cilju povećanja iskoristivosti računarskog sustava kao i skraćenja vremena odziva, više korisničkih i sistemskih programa istovremeno je pohranjeno u memoriji.

Zadaci operacijskog sustava vezani uz upravljanje memorijom mogu se sažeti na:

* praćenje koji memorijski dijelovi su trenutno zauzeti i od koga,
* odlučivanje kada se oslobodi dio memorijskog prostora kojem procesu ga dodijeliti, odnosno koji proces unijeti u tom trenutku u radnu memoriju
* dodjela i oslobađanje memorijskog prostora po potrebi

**Upravljanje sekundarnom memorijom**

Radna memorija može biti nedovoljno velika da prihvati cjelovit program i podatke, te predstavlja medij samo za privremeno skladištenje programa i podataka jer nestankom napajanja sadržaj memorije se briše. Ovi problemi rješavaju se dodavanjem računarskom sustavu sekundarne memorije. Ona je znatno većeg kapaciteta od radne memorije, a podaci su sačuvani i bez prisustva napajanja.

Tako su programi i podaci spremljeni na sekundarnoj memoriji, obično magnetskom disku, a za obradu se unose u radnu memoriju. Također programi tijekom izvođenja učestalo očitavaju podatke s diska, i rezultate ponovo unose na disk. Zbog ovakve učestale uporabe diska efikasnost računarskog sustava direktno je vezana uz samu brzinu ovog uređaja kao i uz efikasnost njegove uporabe.

Operacijski sustav prilikom upravljanja sekundarnom memorijom mora rješavati sljedeće:

* upravljanje slobodnim memorijskim prostorom,
* dodjela memorije,
* upravljanje zahtjevima za pristup sekundarnoj memoriji.

**Upravljanje ulazom/izlazom**

Upravljanje ulazom/izlazom također je jedna od značajki operacijskog sustava koja je dobrim dijelom već razmatrana u prethodnom poglavlju, pa će ovdje biti samo rezimirani osnovni zadaci operacijskog sustava vezanog uz obavljanje ulazno/izlaznih operacija, a to su:

* upravljanje ulazno/izlaznim međuspremnicima i brzom međumemorijom,
* ostvarivanje veze između korisničkog programa i pogonskih programa tzv. (*general device-driver interface*)
* pogonski programi za određene ulazno/izlazne uređaje (*device driver*).

**Upravljanje datotekama**

Zbog jednostavnosti uporabe računarskog sustava, operacijski sustav omogućava jedinstven logički prikaz uskladištenih podataka ili programa. Bez obzira na fizička svojstva medija na kojeg se podaci ili programi skladište, operacijski sustav definira datoteku kao jedinstvenu jedinicu kojoj se pristupa na sekundarnoj memoriji. Operacijski sustav oslobađa korisnika složenog zadatka upravljanja procesom fizičkog zapisa na ili očitanja datoteke sa sekundarne memorije.

Tako se datoteka definira kao skupina povezanih podataka čije značenje određuje stvaralac datoteke. Datoteka može biti program i podaci bilo u izvornom ili izvedbenom obliku, ili samo podaci. Datoteka može biti slobodnog formata npr. tekstualne datoteke ili može biti u određenom formatu. Tako se datoteka sastoji od niza bita, okteta, linija ili zapisa čije značenje određuje tvorac datoteke.

Operacijski sustav primjenjuje apstraktan pristup datotekama, upravljajući sekundarnom. Na logičkoj razini datoteke se organiziraju u sustav direktorija kako bi olakšalo njihovo korištenja. Kako datotekama mogu pristupati korisnici na različite načine (stvaranje, pisanje, čitanje, brisanje, pomicanje) tako operacijski sustav mora imati nadzor tko i na koji način koristi određenu datoteku.

Operacijski sustav zadužen je u osnovi za sljedeće aktivnosti s datotekama:

* stvaranje i uništavanje datoteka,
* stvaranje i uništavanje direktorija,
* osnovne operacije s datotekama,
* fizičko smještanje datoteka na sekundarnu memoriju,
* sigurnosno spremanje datoteka na neizbrisive medije (*backup*).

**Zaštita dijelova računarskog sustava**

Kako računarski sustavi dozvoljavaju istovremeno izvođenje više procesa potrebno je ostvariti zaštitu svakog procesa od nenamjernog ili namjernog neovlaštenog djelovanja drugog procesa. U tu svrhu ostvareni su različiti mehanizmi zaštite dijelova memorije, spremnika procesora, datoteka i drugih resursa računarskog sustava koje koriste procesi. Tako svakom pojedinom korisničkom procesu operacijski sustav dodjeljuje određena prava na korištenje resursa računarskog sustava te nadzire proces tijekom izvođenja kako ne bi povrijedio dodijeljena mu prava.

Tako npr. korisnički program može zahtijevati zapis na neku od memorijskih lokacija u kojoj je upisana naredba ili čak cijeli dio operacijskog sustava. Ukoliko se dozvoli takva operacija, operacijski sustav više ne bi mogao ispravno djelovati.. Slično se može desiti sa sistemskim datotekama na disku koje korisnik može neovlašteno izbrisati ili izmijeniti. I ovakve slučajeve nadzire operacijski sustav. Moglo bi se navesti još mnoštvo sličnih primjera koji ukazuju na značaj dijela operacijskog sustava koji je zadužen za zaštitu.

**Otkrivanje pogrešaka**

Operacijski sustav mora stalno nadzirati sklopovlje računala u cilju otkrivanja pogrešaka u njihovom radu kao i mogućih kvarova. Tako npr. pogreške će nastupiti ako se otkrije da ne odgovara paritet pročitanog podatka iz memorije, ili podatka prenesenog s diska, ako se ustanovi da je izgubljena veza s mrežom, ako je pisač ostao bez papira ili je odspojen, ako je obavljena aritmetička operacija s nedozvoljenim operandima ili se pristupa nedozvoljenoj ili nepostojećoj memorijskoj lokaciji. Za svaki tip pogreške operacijski sustav mora poduzeti određene radnje kako bi osigurao ispravan nastavak rada računarskog sustava.

**Tumačenje upravljačkih naredbi**

Jedna od značajki operacijskog sustava je da ostvaruje vezu između korisnika i računala. Korisni unosi naredbe računalu, koje operacijski sustav interpretira i poduzima određene aktivnosti. Zaseban dio operacijskog sustav (*Command interpreter*) zadužen je za tumačenje naredbi te pokretanje odgovarajućih aktivnosti.

Način na koji korisnik zadaje naredbe razlikuje se od sustava do sustava. Tako npr. MS DOS koristio je linijski tumač naredbi, (*command-line interpreter*) gdje se naredna učitavala s tipkovnice. Korisnički orijentirani sustavi razvili su grafička sučelja kako bi korisniku olakšali proces zadavanja naredbi računalu, pa korisnik komunicira s računalom preko niza ikona, prozora, izbornika.

Upravljačke naredbe koje korisnik zadaje računalu odnose se na stvaranje i upravljanje procesima, obavljanje ulazno/izlaznih poslova, upravljanje glavnom i sekundarnom memorijom, pristup datotekama, zaštita i mrežni rad.

**Umrežavanje (*Networking*)**

Distribuirani sustavi se sastoje od više procesora (računala) koji ne dijele zajedničku memoriju ili zajednički takt. Nasuprot, svaki procesor ima svoju lokalnu memoriju i međusobno komuniciraju preko različitih komunikacijskih kanala (Ethernet, modemski i sl.). Ovakva računala znatno se razlikuju u karakteristikama i namjeni. Mogu biti osobna računala, radne stanice ili velika računala (*mainframe*).

Računala (procesori) u ovakvim sustavima povezani su komunikacijskom mrežom koja može biti konfigurirana na različite načine. Projektant mreže projektira različite mrežne konfiguracije, strategije usmjeravanja informacija (*routing*), zaštite informacija i sl.

Distribuirani sustavi objedinjuju fizički odvojene i po svojstvima različite računarske sustave u jedinstven sustav, osiguravajući korisniku pristup raznovrsnim resursima i informacijama. Pristup zajedničkim resursima ne samo da povećava procesnu moć koja stoji korisniku na raspolaganju, nego osigurava pristup i obradu neizmjerno većeg broja podataka. Operacijski sustav obično osigurava pristup mreži u obliku pristupa datotekama (*file access*), kao i detalje vezane uz umrežavanje u vidu programa za upravljanje mrežnim međusklopovima (*network interface’s device driver*).

# PROCESI

## Koncept procesa

Program koji se izvodi sa diska se upisuje u memoriju, programsko brojilo postavlja se na adresu prve naredbe programa te se pokreće izvođenje. Sa adrese na koju pokazuje programsko brojilo dohvati se naredba i upiše u instrukcijski registar. Zatim se naredba izvede. Izvođenje naredbe rezultira promjenom stanja resursa sustava, konkretno u ovom primjeru stanja programskog brojila i registra R0.





*Slika 3.1.* Izvođenje naredbe na procesoru.

Nakon izvođenja četiri naredbe mijenja se ne samo stanje registara nego i memorijskih lokacija.



*Slika 3.2.* Stanje sustava nakon izvođenja četiri naredbe.

Temeljem navedenih razmatranja u operacijskim sustavima pojam **procesa**. Proces je program u izvođenju čije stanje je u svakom trenutku određeno stanjem resursa sustava, registara i memorijskih lokacija. Proces postaje jedinica obrade u suvremenim računarskim sustavima s vremenskom podjelom poslova.

Proces je, kao što je već napomenuto, program u izvođenju. Procesor izvodi proces izvodeći slijedno naredbu po naredbu procesa. Proces je znatno više od programskog koda koji se izvodi. On obuhvaća i sve trenutne aktivnosti u sustavu opisane sadržajima spremnika procesora i sadržajima memorijskih lokacija koje koristi proces. Tako proces općenito sadrži programski odsječak ili nepromjenjivi tekstualni segment, stog procesa koji sadrži privremene podatke potrebne procesu (podatke koje se prenose u podprogram, povratnu adresu iz podprograma i trenutne varijable) i globalne podatke pohranjene u dio podatkovne memorije (*data section*).

Važno je napomenuti da program sam za sebe nije proces. On je pasivna struktura pohranjena kao datoteka u sekundarnoj memoriji (disku). Proces je aktivni entitet, odnosno program u izvođenju s programskim brojilom koje pokazuje na sljedeću naredbu procesa i pripadajućim skupom spremnika i memorijskih lokacija kao i resursa računarskog sustava koje koristi.

*vektor spremnika* **r** = {r1, r2, ... , ri}

*vektor memorije* **m** = {m1, m2, ... , mj}

Skup **S** = {**r**, **m**} opisuje u svakom trenutku stanje procesa. Proces izvođenje započinje s nekim početnim stanjem **S0** = {**r0**, **m0**} koji sadrži ulazne podatke, propagira se kroz niz stanja prema konačnom stanju **Sn** = {**rn**, **mn**} koji sadrži izlazne podatke.

**S0**, **S1**, **S2**, ... , **Sk**, **Sk+1**, ... , **Sn-1**, **Sn**

Indeksi simboliziraju protjecanje diskretnog vremenskog intervala, odnosno može se govoriti o diskretnom slijedu diskretnih vremenskih stanja kroz koje prolazi proces, a s njime i računarski sustav.

Standardni tipovi računala imaju svojstvo da je stanje k+1 trenutku funkcija stanja u k-tom:

**Sk+1** = fk(**Sk**).

Tako je za svaki proces unaprijed definiran i niz funkcija:

f0, f1, f2, ... , fk, fk+1, ... , fn-1

Ovo svojstvo omogućava da se proces može u svakom trenutku prekinuti i nakon toga nastaviti ukoliko se sačuva stanje procesa u trenutku prekida **Sk**. Prebacivanje izvođenja s procesa na proces koje ima za posljedicu promjenu sadržaja spremnika procesora naziva se promjena konteksta. Važno je ustanoviti uvjete kada je neki proces moguće prekinuti, te kada ga je moguće ispravno nastaviti.

Problem je odrediti kada je moguće neki proces pi prekinuti i zamijeniti ga procesom pj, s mogućnošću da se proces pi naknadno nesmetano nastavi. Svaki proces tijekom izvođenja koristi određene spremnike i određene memorijske lokacije. Tako neka:

proces pi koristi **ri** i **mi**,

proces pj koristi **rj** i **mj**.

Sadržaje spremnika moguće je relativno jednostavno sačuvati pohranjivanjem na stog, ali sadržaje memorijskih lokacija nije moguće jednostavno prebacivati. Zato se može na prvi pogled postaviti zahtjev da dva procesa ne koriste iste zajedničke memorijske lokacije, odnosno da vrijedi:

**mi** ∩ **mj** = ∅,

Ovakvi procesi nazivaju se nezavisni. Ovaj uvjet nezavisnosti je previše je strog, odnosno moguće je postaviti i nešto blaže uvjete. Naime svaki proces koristi neke memorijske lokacije da bi iz njih čitao i u njih zapisivao ili samo zapisivao, te memorijske lokacije iz kojih samo čita. Neka je:

za proces pi Di – domena i-tog procesa {lokacije iz kojih proces samo čita},

Ri – kodomena i-tog proces {lokacije u koje proces upisuje},

za proces pj Dj – domena j-tog procesa {lokacije iz kojih proces samo čita},

Rj – kodomena j-tog proces {lokacije u koje proces upisuje},

Uvjet nezavisnosti dvaju procesa je:

Di ∩ Rj = Dj ∩ Ri = Ri ∩ Rj = ∅.

Ovo je tzv. **Bernsteinov** uvjet nezavisnosti dvaju procesa. On nezavisne procese definira kao procese koji nemaju zajedničke memorijske lokacije čiji sadržaj se tijekom izvođenja mijenja. Nezavisni procesi mogu čitati iz zajedničkih memorijskih lokacija.

## Prebacivanje izvođenja s procesa na proces

Pretpostavimo da želimo nakon četiri naredbe prvog procesa, prebaciti izvođenje na drugi proces. Drugi proces pretpostavlja inicijalno stanje spremnika, svi spremnici imaju sadržaj 0, a programsko brojilo pokazuje na prvu naredbu procesa, 5000.



*Slika 3.3.* Stanje sustava prije prebacivanja na drugi proces.

Ali nije samo ovo dovoljno napraviti. Želimo da nakon izvođenja određenog broja naredbi drugog procesa prebaci se izvođenje ponovo na prvi proces. On očekuje da nastavi sa stanjem koje je zatekao prije nego se izvođenje prebacilo na drugi proces (Slika 3.2). Zato operacijski sustav mora prije prebacivanja na drugi proces zapamtiti stanje procesa koji se prekida. Budući su procesi nezavisni ne mijenjaju sadržaje memorijskih lokacija drugih procesa dovoljno je zapamtiti za proces koji se prekida sadržaje registara procesora.

### Deskriptor procesa (*Process Control Block*)

Operacijski sustav mora na neki način pratiti stanje procesa, te učinkovito mijenjati stanja procesa. U tu svrhu svaki proces prikazan je **deskriptorom procesa** (*Process Control Block*). Deskriptor procesa je tablica u koju su uneseni svi značajni podaci o procesu:

* **pokazivači** koje operacijski sustav koristi kako bi preko PCBa upravljao procesom,
* **stanje procesa** (proces može biti novi, pripravan, aktivan čeka ili je zaustavljen, završen),
* **ime procesa i informacije za obračun**, operacijski sustav ime informacije o nazivu procesa, kome pripada proces, koliko vremena je koristio procesor, koliko memorije je koristi te ostale podatke potrebne za obračun,
* **sadržaj programskog brojila** koji pokazuje na sljedeću naredbu procesa koju je potrebno izvesti,
* **sadržaji spremnika procesora** koji moraju biti sačuvani kako bi se proces kad ponovo dobije pravo korištenja procesora mogao nesmetano nastaviti,
* **prioriteti procesa** su podaci koje operacijski sustav koristi kako bi od svih procesa koji su pripravni za izvođenje mogao učinkovito odabrati kojem procesu dodijeliti pravo korištenja procesora,
* **podaci za upravljanje memorijom**, kako je više procesa prisutno u radnoj memoriji potrebno je paziti koji dijelovi memorije pripadaju pojedinom procesu,
* **popis datoteka koje proces je otvorio**,
* **popis resursa sustava koji su potrebni procesu.**

Veličina i raspored informacija koje se unose u deskriptor procesa variraju u ovisnosti o operacijskom sustavu. Operacijski sustav ove podatke memorira u obliku tablice:

|  |  |
| --- | --- |
| pokazivači | stanje procesa |
| naziv procesa | |
| podaci za obračun | |
| programsko brojilo | |
| spremnici | |
| podaci o memoriji koju proces koristi | |
| popis otvorenih datoteka | |
| popis resursa  koje proces koristi | |
| •  • | |

*Slika 3.4. Izgled deskriptora procesa.*

### Stanje procesa

Tijekom izvođenja procesa on mijenja svoja stanja. Stanje procesa određeno je njegovom trenutnom aktivnošću. Tako kada korisnik ili operacijski sustav namjerava izvesti neku proceduru ili neki program stvara **novi** proces. Novi proces iako može odmah započeti s izvođenjem ne mora odmah dobiti procesor na korištenja. On je **pripravan** i kada je procesor slobodan, te kad on dobije pravo korištenja procesora postaje **aktivan** odnosno koristi procesor. Tijekom izvođenja proces može zahtijevati ulazno/izlaznu operaciju ili može biti jednostavno prekinut od strane operacijskog sustava jer mu je isteklo vrijeme dozvoljeno za korištenje procesora. U prvom slučaju procesor **čeka** na izvođenje ulazno/izlazne operacije, a u drugom slučaju on je već **pripravan** za izvođenje ali samo čeka red za dodjelu procesora. Proces kada obavi svoj zadatak je **završio** s izvođenjem. Može se zaključiti da proces može biti u jednom od sljedećih stanja:

* **novi** (*new*) stanje stvaranja procesa,
* **aktivan** (*running*) proces koji se izvodi,
* **pripravan** (*ready*) proces koji čeka na dodjelu procesora,
* **čeka** (*waiting*) proces koji čeka na neku aktivnost (ulazno/izlaznu operaciju, neki signal i sl.),
* **završio** (*terminated*) proces koji je završio obradu.



*Slika 3.5. Dijagram stanja procesa.*

Nazivi pojedinih stanja procesa su proizvoljni i razlikuju se u različitim operacijskim sustavima, ali procesi u svim operacijskim sustavima mogu biti u navedenim stanjima.

Važno je primijetiti da samo jedan proces može biti aktivan, dok više procesa može biti pripravno ili čekati na obavljanje pojedinih ulazno/izlaznih operacija ili čekati na neki događaj u računarskom sustavu.

## Upravljanje procesima (Process Scheduling)

Cilj višeprogramskih računarski sustava je da proces bude praktički cijelo vrijeme zaposlen obradom nekog procesa. U sustavima s vremenskom podjelom korištenja procesora među procesima cilj je što pravednije rasporediti korištenje procesora između više procesa. Kao što je već napomenuto u jednoprocesorskom sustavu uvijek je samo jedan proces aktivan, dok veći broj procesa čeka na dodjelu procesora. Kada jedan proces izgubi pravo korištenja procesora novi pripravni proces se aktivira.



*Slika 3.6. Dijagram prikaza promjene stanja procesora.*

### Redovi procesa

Operacijski sustav sve procese prihvaćene na obradu i koji su pripravni za izvođenje postavlja u red pripravnih procesa organiziran kao red deskriptora procesa. Operacijski sustav kao zaglavlje reda pripravnih sadrži pokazivač na deskriptor prvog i posljednjeg procesa u redu pripravnih procesa.



*Slika 3.7. Red pripravnih procesa.*

Postoje i drugi redovi u sustavu. Proces koji se trenutno izvodi, odnosno onaj kojem je dodijeljen procesor postavlja se u red aktivnih procesa. Praktički u jednoprocesorskom sustavu ovaj red ima samo jedan član. Tijekom izvođenja procesa iz različitih razloga, koji će se naknadno navesti, dolazi do prekida njegovog izvođenja, a na izvođenje se uzima drugi proces.

Osim reda pripravnih postoji i čitav niz drugih redova kao što su redovi za pojedine ulazno/izlazne uređaje tzv. redovi U/I uređaja (*device queue*) organizirani na isti način kao i red pripravnih procesa. U ove redove stavljaju se procesi koji čekaju da budu posluženi na određenom ulazno/izlaznom uređaju. Osim zahtjeva za ulazno/izlaznom operacijom proces može izgubiti pravo korištenja procesora ako stvori novi proces, dijete (*child*) te mora čekati na završetak novostvorenog procesa. Sljedeći slučaj kada neki proces gubi pravo korištenja procesora je kada čeka da se pojavi neki prekid. Procesi koji imaju relativno intenzivnu obradu tj. dug period uporabe procesora mogu biti prekinuti od strane operacijskog sustava i postavljeni u red pripravnih procesa. Moguće situacije koje se javljaju u računarskom sustavu prikazane su na slici 3.5.



*Slika 3.8. Redovi procesa u operacijskom sustavu.*

Prebacivanje procesa iz stanja u stanje obavlja operacijski sustav bilo na način da proces pozove operacijski sustav, odnosno jezgru operacijskog sustava (*kernel*). Pozivi procedura jezgre operacijskog sustava jako su slični prekidima i nazivaju se programski prekidi. Na sličan način procedure jezgre mogu djelovati na proces prebacujući ga u novo stanje.



*Slika 3.6. Postupak prebacivanja procesa iz različitih stanja.*

Sistemski sat jezgri postavlja zahtjeve za prekidom s ciljem da se vodi evidencija koliko dugo je neki proces koristio procesor, te da se procesor pravednije vremenski raspodijeli između procesa.

### Programsko rješenje promjene konteksta

Kako je već opisano operacijski sustav logički organizira PCBove u redove sukladno stanju u kojem se proces nalazi:



*Slika 3.7. Organizacija PCB-ova u redove sukladno stanju procesa.*

Ovakva logička organizacija odgovara fizičkoj organizaciji kako je prikazano sljedećom slikom:



*Slika 3.8. Položaj PCBova u memoriji računala.*

Postupak prebacivanja izvođenja s procesa na proces objasniti će se na sljedećem primjeru:

Neka aktivni proces P1 izvodi ulazno/izlaznu operaciju čitanja znaka s tipkovnice. U tom slučaju proces P1 generira poziv operacijskom sustavu (programski prekid) Operacijski sustav temeljem ove informacije koju mu je proslijedio proces preko prekida obavlja izmjenu konteksta:

1. Pohraniti stanje procesa u njegov PCB. Pokazivač **aktivan** pokazuje na PCB aktivnog procesa pa pohrana stanja procesa izvodi se na način:

*spremi sadržaje spremnika u PCB na koji pokazuje pokazivač aktivan na mjesto rezervirano za registre*

***aktivan->registri++ = reg1;***

***aktivan->registri++ = reg2;***

***…***



*Slika 3.9. Pohrana stanja aktivnog procesa u njegov PCB.*

1. Prebacivanje **aktivnog** procesa (njegovog PCBa) u red **čeka tipkovnicu,** prvog pripravnog procesa u red **aktivan** i ažuriranje reda **pripravan**.

*čekaj\_tipkovnicu = aktivan;*

*aktivan = pripravan;*

*pripravan = pripravan->pokazivač;*

*aktivan = NULL;*









*Slika 3.10. Ažuriranje redova koji opisuju stanja procesa.*

1. Obnova stanja procesora, odnosno sadržaje spremnika iz PCB-a na koji pokazuje **aktivan** (proces 3) prebaci u spremnike procesora.

***reg1 = aktivan->registri++;***

***reg2 = aktivan->registri++;***

***…***



*Slika 3.11. Obnova stanja procesora i aktiviranje procesa 3.*

### Procedure dodjele (Schedulers)

Tijekom izvođenja proces prelazi kroz različita stanja, odnosno biva prebacivan iz jednoga u drugi red. Prebacivanje iz reda aktivnih procesa u neki od redova čekanja izvodi se pozivom sistemskih procedura. Kada proces izgubi pravo korištenja procesora, izbacuje se iz reda aktivnih, a u red aktivnih prebacuje se jedan proces iz reda pripravnih procesa. Kojem procesu iz reda pripravnih dodijeliti pravo korištenja procesora jedan je od zadataka operacijskog sustava. Ovaj algoritam odabira ili dodjele procesora može biti krajnje jednostavan, po principu FIFO, ili nešto složeniji koji će uvažavati karakteristike procesa (prioritet, veličinu i sl.). Kako se očekuje učestala izmjena stanja procesa ovaj algoritam ipak mora biti kratak i relativno jednostavan. On je poznat pod nazivom procedura kratkotrajne dodjele procesora(*short-term schduler*).

U sustavima za skupnu obradu procesa ili sustavima koji pojedine procese obrađuju u pozadini, obično je započet veći broj programa od broja procesa koje sustav može prihvati na obradu, odnosno onih koji se mogu istovremeno obrađivati. Zato se dio tih programa zadržava na disku, a prihvaća se na obradu kada operacijski sustav zaključi da su za to ispunjeni uvjeti, npr. neki proces je završio s obradom. Koji program s diska treba prihvatiti na obradu određuje opet zasebna procedura operacijskog sustava, tzv. dugoročna procedura dodjele (*long-term schduler*). Ova procedura poziva se rjeđe pa može biti složenija.

Pojedini operacijski sustavi uopće ne koriste dugoročne procedure dodjele nego sve započete programe prihvaćaju na obradu. Ovim pristupom ponekad se preopterećuje sustav i previše vremena troši se na upravljanje procesima.

S druge strane neki operacijski sustavi kada uoče da je previše procesa uzeto na obradu pojedine procese deaktiviraju i vračaju ih na disk. Naravno, spomenute aktivnosti zahtijevaju trošenje dodatnog vremena za prebacivanje programa iz memorije na disk i obratno.

### Izmjena konteksta

Prebacivanje obrade s procesa na proces zahtjeva da procesor sačuva stanje prekinutog procesa u njegov deskriptor procesa te da obnovi stanje svojih spremnika iz deskriptora novog procesa. Ova operacija koju izvodi procesor naziva se izmjena konteksta (*context switch*). Kako ova operacija predstavlja dodatan utrošak procesorskog vremena važno je da ona što kraće traje. U tu svrhu mnogi procesori imaju dodatnu sklopovsku podršku s pomoću koje se značajno skraćuje ovo vrijeme. Tako npr. procesor DECSYSTEM-20 ima više grupa spremnika, pa se promjena konteksta obavlja jednostavnim prebacivanjem aktivnosti s jedne na drugu grupu. Naravno ukoliko broj procesa prihvaćenih na obradu je veći od broja grupa spremnika tada je ponovo potrebno koristiti radnu memoriju prilikom izmjene konteksta.

Kod složenih računarskih sustava, složeniji je i operacijski sustav. Jedna od njegovih funkcija je upravljanje radnom memorijom koja se stavlja na raspolaganje procesu. Ova problematika biti će posebno obrađena. Na ovom mjestu potrebno je samo spomenuti da informacije vezane uz dodjelu i korištenje memorije od strane procesa su značajne za njegovo neometano izvođenje te se i one upisuju u deskriptor procesa. Tako, kod izmjene konteksta i ove informacije je potrebno sačuvati. U ovisnosti o složenosti sustava upravljanja memorijom povećava se, i to značajno, količina informacija koju je potrebno sačuvati, a time se dodatno povećava vrijeme izmjene konteksta. Tako izmjena konteksta značajno može utjecati na performanse sustava. U cilju poboljšanja performansi uvode se novi strukture, tzv. niti (*threads*) koje imaju za cilj smanjenje broja izmjena konteksta, što će biti opisano u narednim razmatranjima

# SINKRONIZACIJA MEĐU PROCESIMA

Zavisni procesi definirani su kao procesi koji koriste za pisanje dijelove memorije u koje pišu ili iz kojih čitaju drugi procesi, odnosno čitaju iz memorijskih lokacija u koje pišu drugi procesi. Zavisni procesi mogu dijeliti direktno dijelove memorije (za podatke ili program) ili dijeliti podatke kroz datoteke. Nekontroliran pristup zavisnih procesa zajedničkim podacima rezultira u njihovoj nekonzistentnosti. Zato je važno da operacijski sustav ima mehanizme kojima kontrolira pristup zavisnih procesa zajedničkim podacima.

## Uvodna razmatranja

Primjer zavisnih procesa je problem proizvođača i potrošača poruka. Jedan proces proizvodi a drugi čita ili troši poruke. Neka dva procesa komuniciraju preko spremnika ograničene veličine. U spremnik je moguće pohraniti svega *n* poruka. Da se osigura da proizvođač ne unosi poruku u pun spremnik, odnosno da potrošač očitava poruku iz praznog spremnika uvodi se varijabla *brojač* koja se inicijalizira na vrijednost nula, a povećava se za jedan kako proizvođač unese poruku, odnosno smanji za jedan kada potrošač potroši poruku. Proizvođač i potrošač izvodili bi sljedeće procedure:

**proizvođač**

ponavljaj

proizvedi poruku;

pročitaj *brojač*;

dok je *brojač* = *n*

pročitaj *brojač*;

spremnik[*in*] = poruka;

*in* = *in* + 1;

*brojač* = *brojač* + 1;

do zaustavljanja

**potrošač**

ponavljaj

pročitaj *brojač*;

dok je *brojač* = 0

pročitaj *brojač*;

poruka = spremnik[*out*];

*out* = *out* + 1;

*brojač* = *brojač* - 1;

potroši poruku;

do zaustavljanja

Promatrajući ovako napisane procedura za proizvođača i potrošača one izgledaju sasvim korektne. Ali upitno je kako će se one ponašati ako se izvode na računalu koje istovremeno obrađuje više procesa. Varijabla *brojač* je varijabla koju dijele oba procesa. Neka se promatraju naredbe napisane u višem jeziku za povećanje ili smanjenje vrijednosti brojača, ali pisane u simboličkom, (asemblerskom) obliku:

*brojač* = *brojač* + 1;

spremnik1 = *brojač*;

spremnik1 = spremnik1 + 1;

*brojač* = spremnik1;

*brojač* = *brojač* - 1;

spremnik2 = *brojač*;

spremnik2 = spremnik2 - 1;

*brojač* = spremnik2;

Spremnik1 i spremnik2 mogu biti bilo isti ili različiti spremnici opće namjene ili akumulator. Bez obzira ako ova procesa koriste isti spremnik njegov sadržaj će za određeni proces ostati sačuvan jer ga operacijski sustav pohranjuje u deskriptor procesa tijekom izmjene konteksta. Neka se pretpostavi sljedeći redoslijed izvođenja naredbi ovih dvaju procesa, neka varijabla *brojač* ima trenutnu vrijednost 5 tj. *brojač* = 5:

|  |  |  |  |
| --- | --- | --- | --- |
|  | **izvodi** | **naredba** | **stanje varijabli** |
| 1. | proizvođač | spremnik1 = *brojač*; | spremnik1 = 5 |
| 2. | proizvođač | spremnik1 = spremnik1 + 1; | spremnik1 = 6 |
| 3. | potrošač | spremnik2 = *brojač*; | spremnik2 = 5 |
| 4. | potrošač | spremnik2 = spremnik2 - 1; | spremnik2 = 4 |
| 5. | proizvođač | *brojač* = spremnik1; | *brojač* = 6 |
| 6. | potrošač | *brojač* = spremnik2; | *brojač* = 4 |

Za primijetiti je da ovakvo izvođenje ovih dvaju procesa rezultira u netočnom konačnom stanju varijable *brojač*, koja ima vrijednost 4, *brojač* = 4, umjesto 5. Ukoliko se zamjene koraci 5. i 6. varijabla *brojač* će ponovo imati netočnu vrijednost 6, *brojač* = 6.

Do ovakve situacije došlo je kao posljedica nekontroliranog pristupa varijabli *brojač*, koja je zajednička ovim procesima. Operacijski sustav mora uočiti ovakve situacije i osigurati ispravno izvođenje zavisnih zadataka.

## Kritičan odsječak procesa

Ako se dio procedure koji izvodi proizvođač poruke prilikom upisa spremnik, kao i dio koji izvodi potrošač poruke dok očitava poruku ne mogu prekidati nego se moraju izvesti nesmetano od početka do kraja tada će se uvijek proces proizvodnje i potrošnje poruka ispravno odvijati. Taj dio procesa u kojem prvo pročita zajedničku varijablu, zatim je obrađuje, te upisuje izmijenjenu vrijednost zajedničke varijable naziva se **kritičan odsječak** (*critical section*). Kritičan odsječak općenito se definira kao dio procesa u kojem proces pristupa ili mijenja zajedničke varijable ili datoteke. Za operacijski sustav je bitno da osigura da kada je jedan proces u kritičnom odsječku tada niti jedan drugi zavisan proces ne smije izvoditi svoj kritičan odsječak. Ovo se naziva **međusobno isključivanje** zavisnih procesa. Da bi ovo osigurao, operacijski sustav mora kontrolirati ulazak svakog procesa u svoj kritični odsječak, odnosno proces mora prilikom **ulaska** u kritični odsječak tražiti od operacijskog sustava pravo za tu operaciju.

Rješenje problema kritičnog odsječka mora zadovoljiti sljedeća tri zahtjeva:

1. **Međusobno isključivanje** (*Mutal Exclusion*): Ako jedan proces izvodi svoj kritičan odsječak tada ni jedan drugi proces ne smije izvoditi svoj kritičan odsječak.
2. **Napredovanje** (*Progress*): Ako ni jedan proces ne izvodi svoj kritičan odsječak, a postoje procesi koji čekaju da uđu u svoj kritičan odsječak, tada samo ti procesi mogu učestvovati u donošenju odluke koji proces će dobiti pravo da prvi uđe u kritičan odsječak. Ulazak u taj odsječak ne može biti odgađan unedogled.
3. **Ograničeno čekanje** (*Bounded Waiting*): Mora postojati vremensko ograničenje unutar kojega se mora omogućiti procesu koji čeka da uđe u svoj kritičan odsječak da to učini.

Potrebno je pronaći rješenje kojim će se riješiti problem izvođenja kritičnog odsječka uvažavajući postavljena tri zahtjeva. Prilikom razmatranja pretpostavka je da proces ima sljedeću strukturu:

ponavljaj

*ulazni dio*

**kritičan odsječak**

*izlazni dio*

nekritičan odsječak

do zaustavljanja

## Sklopovska podrška

Određenim sklopovskim rješenjima moguće je značajno pojednostavniti rješenje problema sinkronizacije. Tako npr. problem sinkronizacije može se riješiti ukoliko se zabrani prekidanje tijekom pristupa zajedničkoj varijabli. Ovim je omogućeno da niz naredbi kritičnog odsječka se izvede bez mogućnosti da tada proces izgubi pravo korištenja procesora. Ovo rješenje je primjerno za jednoprocesorski sustav dok kod višeprocesorskih sustava je njegova implementacija dosta složena. Iz navedenog razloga većina procesora podržava posebnu naredbu koja omogućava da se u jednom ciklusu ispita i promjeni sadržaj neke varijable (*Test-and-Set*) ili da se automatski zamjeni sadržaj dvaju memoriiskih lokacija (*Swap*). Naredba *Test-and-Set* u jednom ciklusu ispituje sadržaj varijable i istovremeno postavlja varijablu u stanje *True*. Ona se može opisati na sljedeći način:

**function** *Test-and-Set* (**var** *zastavica*: boolean): boolean;

**begin**

*Test-and-Set* := *zastavica;*

*zastavica* := *True;*

**end**;

Naredba *Swap* zamjenjujesadržajdvaju varijabli, a može se opisati na sljedeći način:

**procedure** *Swap* (**var** *a, b:* boolean);

**var** *privremena*: boolean;

**begin**

*privremena* := *a;*

*a* := *b;*

*b* := *privremena;*

**end**;

Ovim naredbama olakšano je rješavanje problema kritičnog odsječka. Tako npr. s naredbom *Test-and-Set* ulazak u kritičan odsječak rješava es na sljedeći način:

ponavljaj

dok je *Test-and-Set* (*zabrana*) čini

nikakva operacija;

**kritičan odsječak**

*zabrana* = *False*;

nekritičan odsječak

do zaustavljanja

Ukoliko je dodjela procesora po algoritmu *FCFS* tada ovaj algoritam zadovoljava sve postavljene zahtjeve za rješavanje problema kritičnog odsječka



Na sličan način rješava se problem potpunog zastoja pomoću naredbe *Swap*:

ponavljaj

*ključ* = *True*

ponavljaj

*Swap* (*zabrana,ključ*);

dok je *ključ* = *True*

**kritičan odsječak**

*zabrana* = *False*;

nekritičan odsječak

do zaustavljanja

Ukoliko je se dodjelu procesora koristi algoritam različit od *FCFS* može doći do situacija da neki proces nepravedno dugo čeka na ulazak u kritični odsječak.



Algoritam s naredbom *Test-and-Set* koji rješava ovaj problem koristi polje logičkih varijabli *čekanje*[i] i logičku varijablu *zabrana*.

**var** *čekanje*[0..*n*-1] boolean;

*zabrana, ključ* boolean;

*j, i* cjelobrojna;

ponavljaj

*čekanje*[i] := *True*;

*ključ* := *True*;

dok je (*čekanje*[i] & *ključ*) = *True* čini

*ključ* := *Test-and-Set*(*zabrana*);

*čekanje*[i] := *False*;

**kritičan odsječak**

*j* := *indeks sljedećeg procesa*;

dok je (*j* ≠ *i* & *čekanje*[*j*] = *False*) čini

j :=indeks *sljedećeg procesa*;

ako je *j* = *i* tada

*zabrana* := *False*;

inače

*čekanje*[j] := *False*;

nekritičan odsječak

do zustavljanja

Ovim algoritmom proces prilikom izlaska iz kritičnog odsječka ispituje ciklički preostale procese (i+1, i+2, ..., n-1, 0, 1, i-1) kako bi ustanovio koji je prvi sljedeći proces koji je postavio zahtjev za ulazak u kritični odsječak (*čekanje*[*j*] := *True*). Ovim postupkom postignuta je pravednost prilikom dodjele prava za ulazak u kritični odsječak.

## SEMAFORI

Problem ulaska procesa u kritičan odsječak, ali i svi drugi problemi vezani uz sinkronizaciju među procesima efikasno se rješava uporabom **semafora**. Semafor **S** je cjelobrojna varijabla kojoj se pristupa pomoću dvije osnovne (*atomic*) operacije **čekaj**  i **postavi**. Operacija **čekaj** (*wait*) izvorno se označava s **P** (od nizozemske riječi *proberen*), a operacija **postavi** (*signal*) s **V** (od nizozemske riječi *verhogen*). Općenito razlikuju se binarni i opći semafori. Opći semafor može poprimiti bilo koju cjelobrojnu vrijednost dok binarni semafor može poprimiti vrijednost 0 ili 1, odnosno on je poseban opći semafor. Za opći semafor vrijede sljedeće osnovne operacije:

**čekaj** (S): dok je S < 0 čini

nikakva operacija;

S := S - 1;

**postavi** (S): S := S + 1;

Važno je napomenuti da operacije izmjene vrijednosti semafora moraju biti nedjeljive. Npr. dok neki proces izvodi operaciju **postavi** (S) koja bi u simboličkom obliku izgledala:

**postavi** (S)

spremnik = S

spremnik = spremnik +1

S = spremnik

cjelokupni postupak prebacivanja vrijednosti semafora u spremnik, inkrementiranje sadržaja spremnika i spremanje sadržaja spremnika u memoriju na lokaciju dodijeljenu semaforu ne smije biti prekinut. Isto vrijedi i za operaciju **čekaj** (S).

Semafori su sastavni dio procedura jezgre operacijskog sustava, pa operacijama nad semaforima direktno upravlja operacijski sustav, a procesi koji zahtijevaju pristup semaforima to izvode pozivom operacijskog sustava.

### Primjena semafora

Semafori se učinkovito koriste za međusobno isključivanje prilikom ulaska procesa u kritičan odsječak. Semafor S inicjalizira se na vrijednost 0.

ponavljaj

**čekaj** (S)

**kritičan odsječak**

**postavi** (S)

nekritičan odsječak

**proces P0**

nekritičan odsječak

**čekaj** (S)

**{kritičan odsječak}**

naredba

prekid

prekid

naredba

**postavi** (S)

nekritičan odsječak

prekid

**proces P1**

nekritičan odsječak

**čekaj** (S)

**čekaj** (S)

**čekaj** (S)

**{kritičan odsječak}**

naredba

**vrijednost S**

0

-1

-1

-1

-1

-1

-1

-1

0

0

-1

-1

-1

Osnovni nedostatak svih do sada analiziranih algoritama za rješavanje problema ulaska u kritičan odsječak je da proces koji nije dobio dozvolu za ulazak u kritičan odsječak aktivno čeka trošeći vrijeme procesora. Kako bi se riješio i ovaj problem potrebno je promijeniti operacije **čekaj** i **postavi**. Proces koji izvodi operaciju **čekaj** i zaključi da ne smije nastaviti izvođenje mora sam prekinuti svoje izvođenje i stati u red čekanja na semaforu. Potom se izvođenje predaje operacijskom sustavu koji aktivira neki proces iz reda aktivnih procesa. Ovim pristupom uvode se dodatni redovi čekanja na semaforima. Naime uz svaki oblik kritičnih odsječaka vezan je jedan semafor koji ima svoj red čekanja.

Jedan od procesa koji čeka u redu na neki od semafora mora biti aktiviran, odnosno iz reda čekanja prebačen u red pripravnih procesa, od strane procesa koji izvodi operaciju **postavi** na tom semaforu.

Kako bi se mogle realizirati opisane operacije semafor se definira kao zapis:

**zapis** *semafor*

*vrijednost:* cjelobrojna;

*L*: lista procesa

**kraj**

Svaki semafor ima svoju vrijednost i listu procesa koji čekaju na taj semafor. Kad proces izvodi operaciju **čekaj** na tom semaforu, a semafor nije prolazan, stavlja se u red na taj semafor, odnosno dodaje se u listu procesa u zapisu semafor. Izvođenje operacije **postavi** prebacuje prvi proces iz liste procesa koji čekaju u red pripravnih procesa. Operacije nad semaforima definiraju se sada na sljedeći način:

**čekaj** (S):

S.vrijednost := S.vrijednost – 1;

ako je S.vrijednost < 0 tada

stavi deskriptor procesa u S.L;

**aktiviraj** proces iz reda PRIPRAVAN;

**postavi** (S):

S.vrijednost := S.vrijednost + 1;

ako je S.vrijednost ≤ 0 tada

odaberi jedan deskriptor procesa iz S.L i stavi ga u red PRIPRAVAN;

Procedura **aktiviraj** podrazumijeva poziv algoritma za dodjelu procesa i obnavljanje sadržaja spremnika procesora iz deskriptora procesora koji se aktivira. Koji proces iz reda procesa koji čekaju na semaforu će biti prebačen u red pripravnih procesa ovisi o algoritmu odabira koji u najjednostavnijem slučaju može *FCFS*.

Operacije nad semaforima su kritične operacije pa se mora osigurati da samo jedan proces može izvoditi ili operaciju **čekaj**(S) ili **postavi**(S). Ovo se u jednoprocesorskom sustavu rješava na način da se zabranjuje prekidanje dok se izvode ove operacije. Tako aktivan proces aktivira jednu od navedenih operacije određenim sistemskim pozivom, koji prvo zabrani prekidanje, a potom izvede traženu operaciju. Po završetku sistemskih poziva uobičajeno da aktivni proces izgubi pravo korištenja procesora.

### Binarni semafor

Prethodno opisani semafori nazivaju se opći semafori ili semafori s brojanjem. Za razliko od ovog tipa semafora postoje i binarni semafori koji poprimaju vrijednost 0 ili 1. Kada je vrijednost binarnog semafora 1 semafor se smatra prolaznim, a kada je njegova vrijednost 0 on je zatvoren.

**čekaj** (BSj):

ako je BSj = 0 tada

prebaci deskriptor procesa u red za BSj;

inače

BSj := 0;

**postavi** (BSj):

ako je red BSj prazan tada

BSj := 1;

inače

prebaci proces iz reda BSj u red PRIPRAVAN;

Operacije nad binarnim semaforima su jednostavnije od onih nad općim, pa je zato gdje je god to moguće dobro problem rješavati pomoću binarnog semafora. Tako se binarni semafor može koristiti za međusobno isključivanje.

ponavljaj

**čekaj** (BSj)

**kritični odsječak**

**postavi** (BSj)

nekritični odsječak

do zaustavljanja

### Potpuni zastoj i izgladnjivanje

Primjenom semafora u operacijskim sustavima učinkovito je riješen problem kritičnog odsječka, ali ima za posljedicu pojavu da se procesi mogu međusobno blokirati. Ovaj problem najjednostavnije je prikazati na primjeru dva procesa P0 i P1 i dva semafora S i Q. Neka procesi izvode sljedeće:

**proces P0**

**čekaj** (S);

**čekaj** (Q);

kritičan odsječak S

kritičan odsječak Q

**postavi** (S);

**postavi**(Q);

**proces P1**

**čekaj** (Q);

**čekaj** (S);

kritičan odsječak Q

kritičan odsječak S

**postavi** (S);

**postavi**(Q);

Neka se izvođenje odvija na sljedećim redoslijedom:

**proces P0**

**čekaj** (S);

prekid

**čekaj** (Q);

prekid

**proces P1**

**čekaj** (Q);

**čekaj** (S);

Nakon ovog slijeda operacija proces P0 je u redu čekanja na semafor Q, a proces P1 u redu čekanja na semafor S. Uvjet ponovnog aktiviranja procesa P0 je da proces P1 izvede operaciju **postavi** (Q), odnosno uvjet da P1 može ponovo nastaviti izvođenje je da proces P0 izvede operaciju **postavi** (S). Kako navedene operacije ne mogu biti izvedene procesi P0 i P1 su blokirani, a ovakva situacija se naziva **potpuni zastoj** (*Deadlock*).

prekid

Kaže se da je skup procesa u stanju potpunog zastoja kada svaki proces iz ovoga skupa čeka na događaj koji može izvesti samo neki proces iz istog skupa. Događaji koji su obično vezani uz ovu pojavu su zahtjev za dodjelom nekog resursa i oslobađanje istog. U računarskom sustavu postoji niz situacija koje mogu rezultirati potpunim zastojem, a one će biti naknadno obrađene.

Problem sličan problemu potpunog zastoje je **izgladnjivanje** (*starvation*). Ovo je situacija kada proces zbog nepravednog algoritma dodjele resursa čeka beskonačno u red na taj resurs.

#### PRIMJERI SINKRONIZACIJE

U sljedećem dijelu obraditi će se nekoliko tipičnih primjera kada je potrebno izvoditi sinkronizaciju među procesima u kojima će se ovaj problem rješavati pomoću semafora.

**Problem ograničenog spremnika**

Ovaj problem, koji je već spominjan, uobičajeno se koristi kako bi se prikazalo djelovanje algoritma za sinkronizaciju. Proizvođač proizvodi poruke i sprema ih u ograničeni spremnik dužine *n* poruka. Potrošač iz istog spremnika troši poruke. Sinkronizacija ovih dvaju procesa (proizvođača i potrošača) odvija se pomoću sljedećih semafora: *pristup*, *pun* i *prazan*. Semafor *pristup* osigurava da samo jedan proces ima pristup zajedničkom spremniku, a semafori *pun* i *prazan* broje broj unesenih poruka, odnosno broj praznih mjesta. Semafor *pristup* je binarni semafor inicijaliziran na vrijednost 1 (prolazan), a semafori *pun* i *prazan* su opći semafori inicijalizirani na vrijednost 0 i *n*. Proizvođač i potrošač izvode sljedeće procedure:

**proizvođač**

ponavljaj

proizvedi poruku;

**čekaj**(*prazan*);

**čekaj**(*pristup*);

unesi poruku u spremnik;

**postavi**(*pristup*);

**postavi**(*pun*);

do zaustavljanja

**proizvođač**

ponavljaj

**čekaj**(*pun*);

**čekaj**(*pristup*);

izvadi poruku iz spremnik;

**postavi**(*pristup*);

**postavi**(*prazan*);

potroši poruku;

do zaustavljanja

Analizom ovih procedura uočava se da proizvođač prvo ispituje da li ima mjesta u spremniku za poruku, ako ima smanjuje broj praznih mjesta, zatim ispituje može li se pristupiti spremniku, a tek onda unosi poruku u spremnik, dozvoljava pristup spremniku te povećava brojač poruka. Procedura koju izvodi potrošač poruka potpuno je simetrična. Prvo se ispituje da li je spremnik prazan, ako nije smanjuje se broj poruka u spremniku, zatim se ispituje može li se pristupiti spremniku, a tek onda se vadi poruka iz spremnika, ponovo dozvoljava pristup spremniku te povećava brojač praznih mjesta.

**Problem čitanja - pisanja (*Readers – Writers Problem*)**

U računarskim sustavima sasvim je uobičajeno da se neku datoteku ili zapis dijeli više programa ili više procesa. Neki procesi samo zapisuju u datoteku, a drugi samo čitaju podatke iz nje. Tako se razlikuju procesi nazvani *čitači* (*readers*) i procesi nazvani *pisači* (*writers*). Procesi koji samo čitaju iz datoteke njoj mogu pristupati istovremeno, ali ako neki proces upisuje u datoteku tada svi ostali procesi koji namjeravaju koristiti datoteku moraju čekati da se upis u datoteku završi. Kako bi se riješio navedeni problem zahtjeva se *pisač* ima ekskluzivno pravo pristupa zajedničkom objektu. Navedeni problem naziva se problem **čitanja – pisanja** (*readers – writers*). Ima više varijanti rješenja ovog problema, a sva uključuju rješavanje problema prema određenim prioritetima.

Najjednostavniji algoritam je da *čitač* ima prioritet nad *pisačem*. Čitač odmah pristupa zajedničkom objektu ukoliko već njemu nije dobio pravo pristupa neki *pisač*. Drugim riječima, niti jedan *čitač* ne smije čekati zato što *pisač* već čeka. Sljedeći algoritam ima rješenje da *čitač* ima prioritet prilikom pristupa zajedničkom objektu, odnosno obavlja operaciju zapisa što je prije moguće. Za primijetiti je do oba algoritma postoji problem  *izgladnjivanja* nekih procesa. U prvom algoritmu to su *pisači,* a u drugom *čitači.*

U prvom algoritmu za rješenje problema čitač-pisač, *čitač* koristi sljedeće podatke:

*pristup, upis*: semafor;

*brojilo*: integer;

Semafori *pristup* i *upis* inicijaliziraju se na vrijednost 1, a varijabla *brojač* na 0. Semafor *upis* zajednički je *čitačima* i *pisačima*. Semafor *pristup* koristi se prilikom pristupa zajedničkoj varijabli *brojač* koja broji koliko trenutno *čitača* čita iz zajedničkog objekta. Semafor *upis* koristi se za međusobno isključivanje *pisača*, ali ga ne koriste *čitači* koji ulaze ili izlaze iz kritičnog odsječka dok postoje drugi *čitači* koji su u kritičnom odsječku. Procesi *čitač* i *pisač* izvode sljedeće procedure:

***pisač***

*čekaj*(*upis*);

upiši u zajednički objekt;

*postavi*(*upis*);

***čitač***

*čekaj*(*pristup*);

*brojač* := *brojač* + 1;

ako je *brojač* = 1 tada

*čekaj*(*upis*);

*postavi*(*pristup*);

čitaj iz zajedničkog objekta;

*čekaj*(*pristup*);

*brojač* := *brojač* – 1;

ako je *brojač* = 0 tada

*postavi*(*upis*);

*postavi*(*pristup*);

Za primijetiti je da ako *pisač* izvodi kritični odsječak, a *n* *čitača* čeka na izvođenje kritičnog odsječka, samo prvi *čitač* se stavlja u red na semafor *upis*, a svi ostali (*n* – 1) u red na semafor *pristup*. Ako postoje procesi *pisač* koji su u kritičnom odsječku, semafor *upis* je neprolazan pa proces *čitač* ne može ući u kritičan odsječak.

#### Problem pet filozofa

Pet filozofa provode život razmišljajući i jedući. Oni dijele zajednički okrugli stol s pet stolica, pet tanjura i pet štapića za rižu, a na sredini stola je zdjela s rižom. Kada filozofi razmišljaju tada su nezavisni od svojih kolega, ali kada ogladne tada moraju prvo sjesti, uzeti lijevi štapić, zatim desni štapić i tek onda započeti jesti. Princip je da se smije uzeti odjednom samo jedan štapić. Kada završi jesti filozof odloži lijevi štapić, zatim desni štapić pa tek onda odstupi sa stola.



Ovaj problem smatra se jedan od klasičnih problem sinkronizacije ne zato što pripada skupini problema koje je potrebno rješavati u računarskim sustavima nego zato što zorno opisuje princip upravljanja zavisnim procesima. U računarskim sustavima uobičajena je potreba raspodjele resursa između više procesa vodeći računa da ne dolazi do potpunog zastoja ili izgladnjivanja.

Jednostavno rješenje ovog problema bilo bi uvođenje semafora za svaki štapić. Svaki filozof uzima štapić izvođenjem *čekaj* operacije nad semaforom tog štapića, a odlaže štapić izvođenjem operacije *postavi* nad semaforom tog štapića.

ponavljaj

*čekaj*(*štapić*[i]);

*čekaj*(*štapić*[i+1 **mod** 5]);

jedi;

*postavi*(*štapić*[i]);

*postavi*(*štapić*[i+1 **mod** 5]);

misli;

do zaustavljanja

Iako ovo rješenje osigurava da ne mogu dva susjeda istovremeno jesti, nedostatak je ukoliko svih pet filozofa istovremeno pristupe stolu. Tada će svaki filozof izvođenjem operacije *čekaj*(*štapić*[i]) uzeti po jedan štapić te pokušati uzeti drugi štapić koji je već rezervirao njegov susjed. Ni jedan filozof tako ne može jesti, odnosno procesi se međusobno blokiraju. Tako je došlo do potpunog zastoja.

Postoji više postupaka kako rješavati problem potpunog zastoja. Npr. za ovaj problem moguće je ovaj problem riješiti sljedećim uvjetima:

* dozvoliti da maksimalno četiri filozofa pristupe istovremeno stolu,
* dozvoliti filozofu da uzme štapić samo ako su oba štapića slobodna (uzimanje štapića spada u kritičan odsječak),
* primijeniti nesimetrično rješenje da npr. da filozof koji sjeda na neparnu stolicu uzima prvo lijevi, a zatim desni štapić, dok filozof koji sjeda na parnu stolicu prvo uzima desni, a zatim lijevi štapić.

Naravno u svim mogućim rješenjima ovog problema potrebno je voditi računa da ne dolazi do izgladnjivanja jednog od filozofa. Naime, rješavanje problema potpunog zastoja ne uključuje i rješenje problema izgladnjivanja.