PREGLED ISTORIJATA OPERATIVNIH SISTEMA-Prva generacija operativnih sistema (posle I svetskog rata): -nema klasicnog operativnog sistema, -ista grupa ljudi je projektovala, pravila, programirala i odrzavala racunar. Druga generacija operativnih sistema (1955 - 65): -pojavili su se tranzistori, bušene kartice, trake, -paketna obrada (batch), -programski jezik fortran. Treca generacija operativnih sistema (1965 - 80): -IBM 360 - tada najpopularnija mašina, -primena integrisanih kola, -pocetak pravljenja operativnih sistema ali su pravljeni nesistematski, -multiprogramiranje – omogucava da nekoliko programa istovremeno ili konkurentno koriste racunarske resurse (izvrsavanje vise programa u istom intervalu vremena).

RASPODELA VREMENA-Korisnici sede za terminalima. Računar u jednom malom vremenskom intervalu radi sa jednim korisnikom (nekoliko milisekundi), a zatim sa drugim korisnikom... svaki korisnik ima utisak da računar radi samo sa njim. IBM,
Bell
laboratories, General Electric,
Boston
: ove firme su htele da naprave jedan racunarski sistem za ceo grad: ovaj projekat se zvao MULTICS (Multiplexed Information and Computing Service) - propao je.
Ken Thompson, na računaru POP-7 je napravio racunarski sistem za jednu osobu (Uniplexed Information and Computing Service – UNICS => UNIX). Dennis Ritchie je napravio programski jezik C i ova dvojica su zajedno napravili operativni sistem UNIX koji je bio najpopularniji u to vreme. UNIX sistem je i danas na tržištu. UNIX je napravljen pomoću C-a, vrlo je efikasan i ima razne varijante.

Četvrta generacija operativnih sistema (1980 - 90)-personalni racunari (MS DOS). Dos je bio napravljen prakticno slucajno i bio je igracka naspram UNIX-a, Microsoft je zadržao licencu na DOS. WINDOWS – koji je verovatno gori od DOS-a , kasnije su konačno napravili nešto mnogo bolje: WINDOWS NT. Tannenbaum je napravio manju verziju UNIX-a, takozvani MINIX koji je bio potpuno kompatibilan sa UNIX-om i bio je veoma raširen desetak godina. Jedan student (Linus Thornwald) je uzeo minix i preradio ga, napravio je malo ozbiljniju verziju i nazvao ga LINUX koji je zvanicno besplatan (nema licencu) i kompatibilan je sa UNIX-om.

RACUNARSKI SISTEMI-Organizacija racunarskog sistema: Sistemska magistrala prenosi adresne signale, podatke... Kontroleri u sebi imaju procesore. Kontroler za štampač - upravlja radom
štampača; Kontroler za disk - upravlja
radom
diskova... Kod ukljucenja (reset-a) sistema. Uključuje se bootstrap program – početni program,

vrsi inicijalizaciju sistema (napuni registre, pripremi sistem za punjenje operativnog sistema – BIOS, sprema sve za sledeći korak). Punjenje operativnog sistema. Startovanje prvog procesa (programa, informacija, neki sadržaj registara...) koji vrši interakciju sa korisnikom.Operativni sistem upravlja
radom
racunara, čeka u petlji da se nesto dogodi (pritisak na tastaruri, pomeranje miša). Dogadjaj je neki prekid. Prekid može biti: hardverski (tastatura, disk, takt realnog vremena), softverski (instrukcija za prekid (simulira prekid)). Odziv procesora na prekid: procesor prekida izvršavanje tekućeg programa, pokreće se izvršavanje programa za obradu prekida,po završetku vraća se na prekinuti program.
Jedinica koja generise prekid dostavlja procesoru podatak koji se zove vektor prekida. Vektor prekida se koristi kao pokazivac na tabelu sa adresama programa za obradu prekida. Ova tabela se nalazi na pocetku memorije. Za svaku adresu je rezervisano 4 bajta i (kod PC-a) ima 256 adresa vektora prekida. Bez razumevanja prekida, praktično je nemoguće razumeti šta i kako operativni sistem radi. Prekid moze da se zabrani (maskira) ili dozvoli. Operativni sistem radi na principu odziva na prekid (interrupt driver).

ULAZ/IZLAZ-Kontroler: upravlja periferijskom jedinicom, organizuje prenos podataka, sadrzi lokalni bafer za prihvat podataka, je programabilna jedinica (postoje registri koje procesor moze da programira). Princip rada: procesor programira kontroler, kontroler pokrece periferijsku jedinicu, kontroler organizuje prenos podataka, kada se zavrsi prenos generise prekid. Obicno korisnicki program zahteva U/I koji moze biti sinhroni i asinhroni. Sinhroni U/I: korisnicki program zahteva U/I, pocinje izvrsavanje U/I, kada se zavrsi U/I programska kontrola se vraca korisnickom program. Na ovom principu radi MS DOS. Ovakav nacin realizacije U/I nije efikasan. Asinhroni U/I: Kontrola se odmah vraca porisnickom procesu.Ovo je mnogo bolji nacin rada i povecava efikasnost racunarskog sistema. DMA (Direct Memory Access): koristi se za brzi prenos podeteka direktno izmedju periferijske jedinice i memorije, bez DMA kontrolera procesor mora svaku rec prvo da prenese u svoje registre pa onda da je prosledi u memoriju ili periferijsku jedinicu, sa DMA podaci se direktno prenose izmedju memorije i periferijske jedinice bez ucesca procesora, dok traje DMA prenos procesor moze da radi nesto drugo. Memorija: postoje razne vrste memorija, procesor moze da neposredno prisupi samo poluprovodnickoj (operativnoj) memoriji, LOAD – procesor uzina podatak iz memorije i prebaci je u svoje registre, STORE – procesor prebaci podatak iz svojih registara u memoriju. Racunari koje izucavamo su Fon Nojmanovog tipa: instrukcije su sekvencijalno smestne u memoriju i uvodi se registar koji se zove programski brojac koji pokazuje na memorijsku lokaciju u kojoj se nalazi instrukcija koja treba da se izvrsi.

Memorijska hijerarhija: u računarskom sistemu postoje razne memorije. Osobine memorija - Memorije se razlikuju po: brzini pristupa (MAX - rp; MIN - mm), kapacitetu (MAX - mm; MIN - rp), cena (po bitu) (MAX - rp; MIN - mm, volativnosti (da li se sadrzaj memorije gubi kada se iskljuci napajanje - masovna memorija ne gubi sadržaj), načinu pristupa (serijski ili paralelni - operativna i keš memorija, registri procesora), da li procesor moze direktno da pristupi memoriji

(ne može direktno da pristupi masovnoj memoriji).

KEŠIRANJE-Keširanje je postupak prebacivanja delova memorijskog prostora iz memorije sa nizeg u memoriju sa visim hijerarhijskim nivoom. Keširanje povećava efikasnost sistema. Tri vrste keširanja memorije:

ZASTITA RAČUNARSKOG SISTEMA-ne rade se kod jednokorisnickih sistema, kod multiprogramskih sistema vise programa se izvrsavaju konkurentno. Ne želimo da jedan program utiče na izvršavanje drugog programa - operativni sistem treba da spreči ovakve probleme. To se postiže uvođenjem dva režima rada: korisnički, privilegovani (sistemski ili monitorski režim rada, kaže se još i zaštićeni režim). Cilj uvođenja dva režima rada računara je obezbeđivanje zastite racunarskog sistema od namernih ili nenamernih upada u sistem. Ovaj rezim rada mora biti hardverski podrzani, mora spostojati indikatorski registar, koji će sadržati flag u kome se nalazi informacija o režimu rada (trenutnom, u kome se nalazi računar). P=0, korisnički režim rada; P=1, privilegovani režim rada. U k
orisničkom
režimu rada izvršavaju se korisnički programi.; Postoje ogranicenja u pogledu izvrsavanja nekih operacija (npr. I/O operacije se ne ogu izvršiti, ne moze se menjati vrednost indikatora P). Po
pravilu, operativni sistem radi u privilegovanom režimu rada. Ako korisnickom programu treba usluga programa koji radi u provilegovanom rezimu rada onda korisnocki program mora da koristi takozvane
sistemske pozive
. Sistemski poziv je specijalna instrukcija kojom se pokreće neka rutina iz privilegovanog režima rada, to je u stvari softverski prekid. Samo prekidom se moze preci iz korisnickog u privilegovani rezim rada. Kad se desi prekid, procesor automatski prelazi u privilegovani režim rada i onda operativni sistem izvršava operacije koje je zahtevao korisnički program. Operativni sistem se izvršava u privilegovanom režimu rada - tu se mogu izvršiti sve instrukcije.

ZAŠTITA MEMORIJE: svaki program ima svoju memorijsku zonu i ne sme da pristupi lokacijama koje su izvan te zone, ovo se radi tako sto se hardverski podrzava pomocu gornjeg i donjeg limita, ako program generise adresu izvan svoje zone hardver automatski generise TRAP (to je vrsta prekida).

ZAŠTITA PROCESORA-Primer: cilj je spreciti da program koji je u beskonacnoj petlji zakoci celi racunarski sistem. Obicno se koristi u ovakvim slučajevima tzv. timer koji prekida procesor posle isteka nekog vremena. Slično korišćenju tajmera je korišćenje računarskog sistema sa raspodeljenim vremenom (time sharing). Svaki proces dobija interval vremena u kome ima na raspolaganju resurse racunarskog sistema.Kad se desi (sledeći) prekid operativni sistem odabere neki drugi proces i pokrece ga. Da bi se omogućio ovakav nacin rada svaki proces mora da sacuva tzv. kontekst. Kontekst obuhvata sve informacije neophodne da se nastavi proces koji je predhodno bio prekinut. Sam program nije proces. Program u izvršenju je proces. Kod prebacivanja sa jednog procesa na drugi, vrši se promena konteksta. U korisničkom režimu rada mogućnosti su vrlo ograničene (korisnički programi). Sistemski pozivi se "obraćaju" OS-u iz korisničkih programa i hardveru iz OS-a. Hardveru se pristupa tako što korisnički program zahteva od OS pristup hardveru, a to OS izvršava preko nekih rutina. Fini, vrlo komplikovan mehanizam, koji u praksi dobro radi. Postoje standardi koji se prihvataju, ali se teško menjaju.

STRUKTURA OPERATIVNOG SISTEMA-Da bi lakše shvatili OS, mi ga dekomponujemo i onda shvatamo svaku celinu posebno. OS je složen sistem sa puno detalja, mehanizama... Komponente OS-a:

Upravljanje procesima:

Program je deo procesa. Proces je skup svih uslova koje program zahteva. Proces se sastoji od: izvršnog programa, podataka koje program obradjuje, informacije na steku, pokazivača steka, registara procesora... Funkcije OS-a: kreira i briše korisničke i sistemske procese (jedino OS može da napravi proces), suspenduje i aktivira proces, sinhronizuje procese, komunikacija izmedju procesa (dva procesa mogu komunicirati jedino preko OS-a), upravljanje međusobnim blokiranjem.

Upravljanje operativnom memorijom: Da bi procesi mogli da se izvršavaju, oni moraju biti u memoriji. OS vrši: dodelu memorije procesima, oslobađanje memorije, evidentiranje koji delovi memorije se koriste i ko ih koristi, određuje koje procese puni u memoriju, ako ima slobodnog memorijskog prostora.

Upravljanje fajlovima: Fajl je organizovana logička celina sa nekim informacijama. Fajlovi se mogu nalaziti na različitim medijumima. OS: kreira i briše fajlove (jedino OS to može), kreira i briše direktorijume, podržava operacije nad fajlovima i direktorijumima, preslikava (mapira) fajlove na sekundarnu memoriju, vrši back-up fajlova.

Upravljanje komunikacijama: Kod računarskih mreža, OS upravlja (podržava) pristup mreži, pristup fajlovima koji su na drugim računarima.

Zaštita: OS kontroliše pristup programa procesa i korisnika resursima računarskog sistema. Zaštita obuhvata: specifikaciju šta sme da se radi, način sprovođenja ograničenja. Komandni interpreter - program koji ima finkciju interfejsa izmedju korisnika i OS-a. Kad korisnički program ne postoji, korisnički interpreter naše komande "daje" u OS. Korisnički interpreter je, u stvari, ljuska (SHELL). U DOS-u, to su *.BAT fajlovi.

USLUGE OPERATIVNOG SISTEMA- Dele se na: usluge za programere, funkcije (usluge) vezane za računarski sistem. Usluge za programera vez
ane za izvrsavanje programa: punjenje programa u memoriju, izvrsavanje programa, zavrsetak (može biti normalan, ili greška); U/I operacije: pristup periferijskim jedinicama, upravljanje fajlovima (kreiranje, brisanje, upis, citanje); komunikacije: izmedju procesa na jednom racunaru, izmedju procesa preko mreze; detekcija gresaka: greske u procesoru, greske u memoriji, greske u U/I jedinicama, greske u korisnickom programu.
Funkcije koje su vezane za racunarski sistem:
dodela resursa: dodela procesora, dodela memorije; evidencije: koji korisnik i koliko dugo je koristio neke resurse, rade se statisticke obrade, naplata usluga; zastita: cilj: obezbedjenje da se svi resursi koriste na predvidjeni nacin; sigurnost: svaki korisnik mora da potvrdi svoj identitet (lozinka - pasos), sprecavanje korisnika da obavljaju operacije koje nisu dozvoljene, evidentiranje pokusaja neovlascenog upada u racunarski sistem. Sistemski pozivi: to su obicno asemblerske instrukije za softverski prekid, naredbe u programskom jeziku tipa PRINT se prevode u sistemske pozive, prenos parametara kod sistemskih poziva se vrsi: u registrima procesora, u nekoj memorijskoj lokaciji (pointer je u registru), na steku; 5 grupa sistemskih poziva: upravljanje procesima: zavrsetak procesa, nasilni zavrsetak (reset), punjenje u memoriju, izvrsenje, kreiranje i brisanje, uzimanje i postavljanje atributa procesa, cekanje, dodela i oslobadjanje memorije; upravljanje fajlovima: kreiranje i brisanje, otvaranje i zatvaranje, citanje i upis, citanje i postavljanje atributa; upravljanje periferijskom jedinicama: zahtev za pristup, oslobadjanje, citanje i upis, citanje i postavljanje atributa, logicko prikljucenje i iskljucenje periferijske jedinice; upravljanje informacijama: citanje vremena i datuma, postavljanje vremena i datuma, citanje i postavljanje atributa procesa, fajla ili periferijske jedinice; komunikacija: uspostavljanje i prekid komunikacione mreze, slanje i primanje poruka, prenos statusnih informacija.

PROCESI-Proces je program u izvršenju. Da bi se program izvršio, neophodni su: izvršni program, memorijski prostor, podaci koje obrađuje, stek, registri procesora, ostale informacije. Skup svih ovih resursa je proces. Kad se proces privremeno suspenduje, on kasnije mora da se nastavi tačno u istom stanju u kome se nalazio u trenutku suspenzije. Zato se moraju čuvati sve informacije koje su bitne za proces.

Tabela procesa (Process Conltrol Block)-Za svaki proces postoji po jedna tabela procesa. Ove tabele su ulančane u listu procesa. OS periodično suspenduje izvršavanje procesa i odabira neki drugi proces koji će da se izvršava. U trenutku suspenzije, u tabelu se upišu svi podaci vezani za proces. U trenutku aktiviranja procesa iz tabele se uzimaju svi podaci neophodni da se nastavi izvršavanje procesa. OVe dve aktivnosti se nazivaju promena konteksta.

Graf stanja procesa-To je životni ciklus procesa. Jedino OS može praviti proces. Svaki proces ima svoj redni broj. kreiranje - pravljenje procesa, terminiranje - brisanje procesa. Kad je proces spreman, OS ga bira i aktivira. Onda je on u izvršenju. Kad se proces u izvršenju blokira, on čeka na I/O. Kad proces koji je blokiran pređe u stanje "spreman", I/O se završtio. Kad se proces u izvršenju vrati u stanje "spreman", OS bira neki drugi proces za izvršenje. Obično imamo bar 2 liste procesa: lista procesa spremnih za izvršenje, lista blokiranih procesa.

TABELA PROCESA-rukovanje procesom - tu se nalaze: sadržaji registara, stanje procesa, vreme pravljenja procea, vreme korišćenja procesora, identifikacioni broj procesa, prioritet procesa, predak procesa; rukovanje memorijom - tu se nalaze: pokazivač na segment sa kodom (programom), pokazivač na segment sa podacima, limit za pristup memoriji; upravljanje fajlovima - tu se nalaze: korenski (ROOT) direktorijum, radni direktorijum, lista otvorenih fajlova.

OPERACIJE NAD PROCESIMA-Tipična operacija je kreiranje (pravljenje) procesa. Obično jedan proces poziva sistemski poziv koji kreira novi proces. Kad se startuje OS, kreira se jedan jedini proces INIT koji dalje kreira sve ostale. DISPEČIRANJE procesa (SHEDULER) je prebacivanje procesa iz jednog stanja u drugo. Jedino je dispečer nadležan za promenu stanja procesa. TERMINIRANJE procesa - proces se terminira kad sa izvrši poslednja instrukcija; u tom momentu se svi resursi koje je proces držao, oslobađaju (npr. memorija, fajl...). Predak može da terminira proces. Obično se taj sistemski poziv zove ABORT. ABORT je nasilno prekidanje programa. Radi se kad: potomak je prekoračio limite, zadatak potomka više nije potreban. Kad se potomak terminira, obično se terminiraju i njegovi potomci.

NITI (Threads)-Niti su manji procesi koji su u grupi (pripadaju procesu). Nit se sastoji od programskog brojača, registara i steka. Niti u istoj grupi zajednički dele memorijski prostor sa instrukcijama i podacima; resurse (fajlovi, I/O). Stanja niti analogna su stanjima procesa: kreiran, spreman, u izvršenju, blokiran i terminiran.

DISPEČIRANJE-Dispečiranje je proces odabiranja i aktiviranja procesa iz liste procesa koji su spremni za izvršenje. Kriterijumi za ocenu algoritama za dispečiranje: Algorita m mora biti: fer - svaki proces mora doći na red, efikasan - cilj je 100% iskorišćenje procesora, vreme odziva (minimalno), minimizacija vremena čekanja, propusna moć koja se obično meri brojem izvršenih procesa u jedinici vremena. Neki kriterijumi su kontradiktorni. Postupak dispečiranja je složen zbog nedostatka kompletne informacije, procesi najviše rade sa I/O, neki procesi najviše koriste procesor. Postoje dve strategije dispečiranja: dispečiranje sa prekidanjem (PROMPTIVE), dispečiranje bez prekidanja (RETURN TO COMPLETION) - ovo nisu ozbiljni sistemi. Imamo takt realnog vremena koji periodično prekida procesor (aktivira se OS, aktivira se dispečer).

ALGORITAM DISPEČIRANJA PO REDOSLEDU PRISPELIH ZAHTEVA ZA AKTIVIRANJE-K oristi se FIFO lista. Nedostatak: vreme čekanja može biti veoma dugo. Algoritam bez prekidanja je nepogodan za sisteme sa raspodelom vremena. Njegova prednost je što je trivijalan.

ALGORITAM PRVO NAJKRAĆI PROCES-Pogodan je za paketnu obradu. Vreme čekanja se smanjuje.

ALGORITAM KRUŽNO DISPEČIRANJE (ROUND ROBIN)-Jedan od najstarijih algoritama i veoma je jednostavan. Sa prekidanjem. Svaki proces dobija kvantum vremena (10-100ms). Kad istekne kvantum, prosec se suspenduje i aktivira se sledeći proces. Ovaj algoritam je osetljiv na vreme potrebno za promenu konteksta. kvantum: 100ms; vreme promena konteksta. Ako se smanjuje kvantum, onda se procentualno smanjuje iskorišćenje procesora.

ALGORITAM PRIORITETNO DISPEČIRANJE-Algoritam sa prekidanjem. Svaki proces ima nivo prioriteta. Aktivira se proces sa najvišim nivoom prioriteta. problem: dugačak proces sa visokim nivoom prioriteta može neodređeno dugo da zauzima procesor, a to se rešava tako što se kod svakog prekida snižava prioritet programa koji se izvršava.

ALGORITAM SA VIŠESTRUKIM REDOVIMA ČEKANJA-Definišu se klase prioriteta. Cilj je smanjenje vremena za promenu konteksta. Procesima u najvišoj klasi prioriteta se dodeli po 1 kvantum za izvršenje, procesima u sledećoj klasi prioriteta se dodeljuju 2, sledeći 4,8,16... Kad se proces u izvršenju suspenduje, prebacuje se u sledeću, nižu klasu.

SINHRONIZACIJA PROCESA-Procesi moraju medjusobno da saradjuju. Međusovno isključivanje (mutual exclusion). problem je pronači način da najviše jedan proces pristupa nekom zajedničkom resursu. Deo programa kome treba obezbediti međusobno isključivanja zove se **kritična sekcija**. Pristup je dozvoljen samo iz kritične sekcije. Cilj međusobnog isključivanja je da nikad nisu 2 ili više procesa u svojoj kritičnoj sekciji. 4 uslova za dobro rešenje međusobnog isključivanja: 2 procesa ne ogu biti istovremeno u svojim kritičnim sekcijama, rešenje ne sme da zavisi od relativnih brzina procesora ili broja procesora, proces koji je blokiran izvan svoje kritične sekcije ne sme blokirati ostale procese da uđu u svoje kritičnu sekciju, proces ne sme neograničeno čekati da bi ušao u svoju kritičnu sekciju (fer). Svaki proces koji je iskazao nameru da uđe u kritičnu sekciju, on će to i učiniti (u konačnom vremenu).

Rešenje problema međusobnog isključivanja-rešenje je multiprocesni režim rada - ZABRANA PREKIDA. Kad se desi prekid, aktivira se OS i aktivira se dispečer (komponenta OS-a) koji odabira sledeći proces koji će aktivirati. Skelet kritične sekcije: zabrana prekida, telo KS, dozvola prekida. Problemi: nema prekida, pa prema tome računar ne može da opslužiprekide koji mogu da budu bitni, ne radi sat realnog vremena, ako korisnički program pogreši, može se desiti da nikada ne dozvoli prekid. Softversko rešenje: Korišćenje zajedničke promenljive "lock" (brava).

U početku Lock = 0

Lock = 0 → štampač slobodan

Lock = 1 → štampač zauzet

Proces koji hoće da uđe u svoju kritičnu sekciju testira lock.

Lock = 0 → proces upisuje Lock = 1 i ulazi u KS

Lock = 1 → proces čeka dok ne bude Lock = 0

Ne radi, može dovesti do greške. ***Alternativni pristup***: **klackalica** - naizmenično pristupa P1 i P2. Proces P1: while (true)

{

while (turn != 0) ; // čekanje

kriticna sekcija ();

turn=1;

nekriticna sekcija();

}

Proces P2: while (true)

{

while (turn != 1) ; // čekanje

kriticna sekcija ();

turn=0;

nekriticna sekcija();

}

Problem : ni jedan proces ne može 2 ili više puta za redom da uđe u svoje KS. Narušen je 3. uslov. Proces koji je van svoje KS blokira drugi proces da uđe u svoju KS. Naredbe koje proveravaju uslov za ulaženje u KS bespotrebno se ponavljaju dok se uslovi ne ispune : "BUSY WAITING" dobro rešenje bi bilo suspendovati proces koji mora da čeka na ulazak u KS. ***Algoritamsko rešenje***: Holandski matematičar DECKER prvi je pronašao softversko rešenje međusobnog isključivanja. 1981. god. Peterson je razvio jednostavnije rešenje.

#define FALSE 0;

```
#define TRUE 1;

#define N 2;          //broj procesa

int turn;             //broj procesa koji moze uci u KS

int interested [N]; //u pocetku sve je 0

enter_region (proces)                    //broj procesa koji poziva ovu rutinu

{

int other;

other = 1_process;

interested[process]=TRUE;

turn=process;

while (turn==process&&interested[other]==TRUE);
```

BUSY WAITING

Da bi proces ušao u KS prvo mora da izvrši enter_region(svoj_broj).

P1:

enter_region(1);

kriticna_sekcija();

leave_region(process);

{

interested [process] = FALSE;

}

P1: P0:

enter_region(1); enter_region(0);

kriticna_sekcija(); kritična_sekcija();

leave_region(process); leave_region(0);

Ovo je rešenje koje dobro radi. Osnovni problem za ovo rešenje je BUSY WAITING.
HARDVERSKO REŠENJE: rešenje koje se danas koristi u svim procesorima:

Instrukcija TSL

TSL R1, Lock

R1←Lock

Lock→1

Ove dve mikrooperacije su nedeljive u smislu, kad krene izvršavanje, ne može da se desi pekid (uvek se izvrše obe). Svi procesori danas to podržavaju.

ASEMBLER:

```
enter_region:

tsl reg, lock

cmp reg, 0

jnt enter_region

ret

leave_region:

mov lock, 0

ret
```

Na ovaj način je apsolutno garantovano da će samo jedan proces biti u KS. I ovde se javlja problem BUSZ WAITING. Tek je rešenje koje je bazirano na semaforima to prevazišlo.

OSNOVNI PROBLEMI SA STANDARDNIM REŠENJIMA-Ima dva problema: ne radi, busy waiting - proces koji čeka, tj. ništa ne radi. Rešenje (sledeće na redu): blokiranje procesa koji čeka i aktiviranje procesa koji dođu na red. Rešenje ovoga mogu biti funkcije: sleep, wake up. Dve funkcije koje služe za blokiranje procesa koji čekaju. To su primitivne funkcije koji najčešće obezbeđuje OS. **sleep** - sistemski poziv koji suspenduje proces koji ga poziva. Proces ostaje suspendovan sve dok ga ne probudi neki drugi proces.

wake up (id)

- ovo je funkcija za sistemski poziv koji budi suspendovan proces. id - proces (identifikacioni broj procesa koji treba da se rpobudi). Ilustracioni primer ove dve funkcije na primeru proizvođač, potrošač. Ograničenja: proizvođač stavlja neke informacije u bafer, potrošač ih uzima, šta je ta informacija, uopšte nije bitno. proizvođač proizvođač ne može da stavlja nove informacije u pun bafer, potrošač ne može da uzima informacije iz praznog bafera, Rešenje: ako je bafer pun, suspendujemo proizvođača koga će aktivirati potrošač kada uyme jednu informaciju iz punog bafera, ako je bafer prazan, suspendujemo potrošača, njega će probuditi proizvođač kad stavi informaciju u prazan bafer, Promenljive: count - broj podataka u baferu, N - kapacitet bafera, Funkcije: produce_item() - pravi novi podatak, enter_item() - stavlja podatak ili informaciju u bafer, remove_item() - uzima podatak iz bafera, consume_item() - "potroši" podatak, Proizvođač:

```
#define N 100;                  //kapacitet bafera

int count = 100;

proizvodjac()

{

while(TRUE)                     //ponavljaj
```

```
{

produce_item();

        if (count==N) sleep ();                 //spavaj, ako je bafer pun

        enter_item;                              //stavi podatak u bafer

        count=count+1;

        if (count==1) wake up (potrosac);

}

}
```

Potrosac:

```
{

while(TRUE)                 //ponavljaj
```

```
{

if (count==0) sleep ();

        remove_item();

        count=count-1;

        if (count==(N-1)) wake up (proizvodjac);

consume_item;

}

}
```

Ima problema sa ovih rešenjima. Pretpostavka je da je count=0. Ako je bafer prazan, potrošač je pročitao count da vidi da li je 0, u tom trenutku dispečer zaustavlja potrošača i aktivira proizvođača. Proizvođač stavlja podatak u bafer, inkrementira count, vidi da je jedan i zove wake up da probudi potrošača. Ovaj wake up se izvrši u prazno, jer potrošač još nije sespendovan. Posle toga dispečer aktivira potrošača koji se suspenduje i u tom momentu sistem pada, pošto nema nikoga da ga budi. Oba procesa su zauvek suspendovana.

SEMAFORI-Holandski matematičar DIJKSTRA ih je prvi predložio. 1965. godine je nešto slično napravio. Predlaže da se koristi promena zvana semafor sa pravilima: semafor = 0 , proces se suspenduje, semafor <> 0, proces se nastavlja. Predlaže da se koriste dve operacije nad semaforom: ***DOWN(P)*** - izvršava se pre ulaska procesa u kritičnu sekciju; ako je semafor veći od 0, smanjuje se za 1 i proces nastavlja, ako je semafor = 0, proces se

suspenduje. ***UP (V)*** - izvršava se posle izlaska procesa iz kritične sekcije; on inkrementira semafor ; ako je 1 ili više procesa bilo suspendovano, na ovom semaforu, izabere jedan od njih i aktivira ga, i tako omogući da taj proces završi DOWN operaciju. Proces se nikad ne suspenduje na UP operaciji. Ako je početna vrednost semafora =1, onda se on zove **binarni semafor**.
Binarni semafor garantuje međusobno isključivanje procesa pre ulaska u kritičnu sekciju.
VAŽNO:
računarski sistem mora da garantuje da je operacija pristupa semaforu nedeljiva; operacije su nedeljive, tj. sistem ne sme da ih prekine. DOWN: provera vrednosti semafora, promena vrednosti semafora, ako treba, suspenduje proces. UP: inkrementiranje semafora, aktiviranje suspendovanog procesa.
Primer 3 semafora:
full - broj punih mesta u baferu, emptu - broj praznih mesta u baferu, mutex - binarni semafor za medjusobno isključivanje kod pristupa baferu.

```
#define N 100

type def int semaphore                    //tip podataka za semafor

semaphore mutex=1;

semaphore empty=N;

semaphore full=0;

proizvodjac()

{

int item;
```

```
while (TRUE)

{

produce_item(&item);

down (empty);                    //dekrementira

down (mutex);                    //ulaz u KS

enter_item(item);  //KS

up (mutex);                      //izlaz iz KS

up (full);               //inkrementira broj punih mesta

}

}

potrosac()
```

```
{

int item;

while (TRUE)

{

down (full);

down (mutex);

remove_item(&item);          //KS

up (mutex);

up (empty);

consume_item(item);

}

}
```

Empty - broji prazna mesta, ako ih nema, suspenduje poizvodjača, full - broji puna mesta, ako ih nema, suspenduje potrošača.

MONITORI-Nepažljivom primenom semafora, korisnik mojže da blokira ceo sistem. Primer: ce o sistem može da se blokira, ako korisnik umesto "down(empty); down(mutex)" napiše "down(mutex); down(empty)". Ovo blokiranje se naziva

DEADLOCK - blokiranje dva procesa koji čekaju jedan na drugog (samrtni zagrljaj). Proizvođač je blokiran na empty, a potrošač je blokiran na mutex. Autori Hoare i Brinch Hansen su 1974,75. predložili korišćenje MONITORA. Monitor je kolekcija procesura, promenljivih i struktura podataka koje su zajedno grupisane u jedan modul OS-a. Procesi pozivaju procedure iz monitora kada to žele, ali ne mogu da pristupaju internim strukturama podataka. Samo jedan proces može biti aktivan unutar monitora u bilo kom trenutku. O ovome se brine kompajler : kada proces zove monitor, prvo se proverava da li je neki drugi proces aktivan u monitoru. Ako jeste, onda se proces suspenduje. Dok se procedura ne izvrši, u monitoru ni jedan drugi proces ne može da pozove neku drugu proceduru. (unutar monitora može biti aktivan samo jedan proces, o ovome se brine kompajler. Kada proces zove monitor, prvo se proverava da li je neki drugi proces aktivan u monitoru, ako jeste, proces se suspenduje).

RAZMENA PORUKA IZMEDJU PROCESA (messaqe passing)-Do sada je komunikacija izmedju dva procesa bila preko zajedničke memorije. Kod razmene poruka nema zajedničke memorije. Ovde se koriste 2 sistemska poziva: SEND (primalac, &poruka)

- za slanje poruke, RECEIVE (pošiljalac, &poruka)

- za primanje poruke. Problemi koji nastaju kod razmene poruka: procesi koji razmenjuju poruku mogu biti na različitim računarima u mreži, pa se može desiti da poruka bude zagubljena. U ovom slučaju može se napraviti dogovor, da primalac vrati poruku kad primi poruku. Problem: šta ako se potvrda zagubi? Da bi se ovaj problem rešio, obično se šalje redni broj poruke zajedno sa porukom. (jedini problem je što redni broj raste, pa se povremeno ovaj broj

resetuje). Procesi moraju imati svoje ime i obično se uz ime stavlja i mašina npr: procesAna mašini1.

procesBnamašini1.organizacijaA

Poruke se mogu korisiti i za poziv udaljene procedure.Vrlo se često koristi. Problem koji ovde najčešće nastaje: šta ako server otkaže? Rešenje problema: proizvodjač-potrošač promenom razmen porukaČ svi problemi koji mogu da se reše korišćenjem zajedničke memorije mogu se rešiti primenom razmene poruka.

```
#define N 100                    /* maksimalan broj poruka

proizvodjac()

{

int item;

message m;

while (true) {

procude_item(&item);        /*proizvodi neki item

receive(potrosac,%m);        /*proverava da li ima prayan okvir, ako nema, okvir, on se blokira
```

```
                    /* ako iam, prelazi na pravljenje poruke

build_message(&m,item);

send(potrosac,&m);

}

potrosac()

{

int item;

message m;

for(i=0;i<N,i++) send (producer,&m);          /* slanje N praynih okvira

while(true) {

receive(producer,&m);    /*prima poruku

extract_item(&m,&item);
```

consume_item(item);

send(producer,&m); /* potvrda da je primio neku poruku, tj. vraca prazan okvir

} Poruka se krece od proizvodjaca, do potrosaca, preko bafera za poruke.

Poruka se stavlja u okvir i salje primaocu. Primaoc uzima poruku i vraca prazan okvir. U ovom slucaju OS obezbedjuje bafer za poruke. Ukoliko nema bafera: "randevous", tj jedini nacin da se preda poruka je da se sastanemo sa onim ko treba da primi poruku i da razmenimo poruku. Sinhronizacioni objekti: semafori, monitori, poruke. svi su podjednako dobri (ako jedan moze da resi problem, mogu i druga dva). Sinhronizacioni mehanizmi testiraju se na tipicnim problemima i ti problemi se nazivaju test primeri. FILOZOFI KOJI VECERAJU: imamo 5 filozofa, 5 tanjira, 5 viljusaka. Filozofu trebaju 2 viljuske da bi mogao da jede spagete. Filozofi samo razmisljaju i jedu. Da bi jeo, filozofu trebaju 2 viljuske, kad se najede, ostavi ih i onda razmislja. Problem: svaki filozof uzme svoju levu viljusku i ceka na desnu. Ovaj problem se naziva izgladnjavanje (STARVATION). Cilj: Napisati program za svakog filozofa tako da svaki od njih jede, misli i da nema gladovanja. Resenje koje ne radi:

#define N 5

philosopher (i)

int i;

{

while (true)

```
{

think();

take_fork(i);

take_fork((i+1)%N);                    /* moduo operacije, posle 5 ide opet 1

eat();

put_fork(i);

put_fork((i+1)%N);

}

}
```

Resenje nije dobro, jer ako svaki filozof uzme svoju levu viljusku, sistem se blokira. Jedno dobro resenje: koristi binarni semafor:

```
down(mutex);
```

take_fork(i);

take_fork((i+1)%N);

eat();

put_fork(i);

put_fork((i+1)%N);

up(mutex);

Na semaforu "mutex" pustate jednog po jednog filozofa. Ogranicenje: samo 1 filozof u datom trenutku moze da jede.

ULAZ-IZLAZ-OS upravlja I/O operacijama, I/O jedinicama. Zadaci OS: upravljanje hardverskim

jedinicama, obrada prekida, obrada gresaka kod I/O operacija, interfejs izmedju I/O podsistema i ostalih jedinica.

}

OS pristupa kontrolerima. Svaki kontroler sadrzi programibilne registre kojima OS pristupa i obavlja operaci upis/citanje. Programabilni registri imaju svoje I/O adrese.

IBM PC:

kontroler	I/O adrese	vektori prekida
tastatura	060-063	9
serijski		
RS232	2f8-2ff	11
paralelni port		
(stampac)	278-27f	15

DMA (direktan pristup memoriji). Postoje kontroleri koji mogu da podrze direktan prenos podataka izmedju memorije i periferijske jedinice. DMA kontroler omogucuje prenos bloka podataka izmedju memorije i periferijske jedinice, cime se postize veca efikasnost racunarskog sistema.

ULAZNO-IZLAZNI SOFTVER-Deo OS koji obavlja operacije vezane za I/O. Osobine I/O softvera: gornji slojevi ne treba da zavise od detalja I/O jedinica, imena I/O jedinica su uniformna i ne zavise od hardverskih detalja, greska se obradjuje sto je moguce blize hardverskim jedinicama. sloj softvera koji je najblizi hardveru treba da obavlja obradu greske, I/O operacije treba da se obavljaju asinhrono, koriscenjem prekida; kod singrone interakcije ne postoji povratna sprega. I/O softver obično započinje neku operaciju i zatim čeka potvrdu da je ta operacija izvršena. I/O softver se blokira i čeka u tom stanu na deblokaciju, tj. kada se desi prekid, on će da deblokira I/O softver. I/O softver mora da resi probleme rada sa I/O jedinicama koje mogu istovremeno da opsluzuju: 1 korisnika, vise korisnika. I/O softver obicno se realizuje u slojevima: I - korisnicki orijentisan softver, II - sistemski softver koji ne zavisi od I/O jedinica, III - drajveri za I/O jedinice, IV - programi (rutine) za obradu prekida. Osnovni cilj I/O softvera je organizovanje I/O softvera u slojeve, tako da niži slojevi sakrivaju detalje vezane za I/O hardverske jedinice, a viši slojevi obezbeđuju dobar, jednostavan i ujednačen (uniforman) interfejs za korisnika. Svaka jedinica mora da ima svoj drajver. Programi za obradu prekida: kada drajver aktivira neku I/O operaciju, drajver se obično suspenduje i čeka da ga neko deblokira, kada se desi prekid, aktivira se program za obradu prekida koji uradi sve što treba da bi se deblokirao drajver, program za obradu prekida deblokira drajver (semafor, monitor, npr. su zaduženi za deblokiranje), drajver dalje radi svoj deo posla. Drajveri: svaka I/O jedinica mora da ima svoj drajver, Kad koji zavisi od detalja vezanih za I/O jedinicu, stavlja se u drajver, drajver pristupa registrima kontrolera i na tan način upravlja i komunicira sa I/O jedinicama, drajver mora da "zna" sve detalje vezane za I/O jedinicu: broj registara, značenje sadržaja registara, redosled operacija, drajver mora da prihvati apstraktne naredbe od viših slojeva i da preduziva sve akcije neophodne za njihovo izvršenje.

KORISNIČKI ORIJENTISANI DEO I/O SOFTVERA-bibliotečne procedure, "spooling" sistem. Bibliotečne procedure (libraries) - Ove procedure se linkuju (povezuju) sa korisničkim programom i izvršavaju se u korisničkom režimu rada. "Spooling" sistem - rešava problem pristupa I/O jedinicama koje ne mogu da opslužuju više od jednog korisnika u datom trenutku. Tipičan predstavnik ovakve klase jedinica je štampač. Korisnički proces prvo štampa u fajl, a onda proces "daemon" štampa iz fajla u štampač. "daemon" je jedini proces koji ima pravo da pristupi štampaču. Spooling sistem može da se korisni tod drugih perifernih jedinica: prenos fajlova preko mreže: spooling direktorijum za mrežu, mrežni "daemon" izabere jedan fajl i prenosi ga preko mreže. Korisnički proces je nadležan za I/O pozive, formatiranje, spooling: I/O softver koji ne zavisi od p.j. - ipravlja imenima, zaštita, baferisanje, dodela resursa, Drajver upravlja pp.j. (upis u registre), provera statusa. Medjusobno blokiranje (DEADLOCK) primer: magnetna traka je velikog kapaciteta, 10Gb, ne može da obavi spooling, rešenje je da se traka dodeli jednom

procesu koji je koristi sve dok mu treba, ploter, dodeli se jednom procesu koji ga koristi sve dok ne završi. Medjusobno blokiranje može da nastupi kod pristupa drugim resursima. Primer: zapisi u bazi podataka. Medjusobno blokiranje nastupa kod pristupa resursima za koje mora da se ispuni medjusobno isključivanje.

Resursi-To su entiteti (objekti) kojima pristupaju procesi. To mogu biti ili hardverske jedinice (traka, štampač i sl.) ili informacije (zapis u BP-a). Zajednička karakteristika: resursu može da pristupi samo jedan proces u datom trenutku. Redosled dogadjaja: 1. proces traži pristup resursu, 2. ako je resurs slobodan, dodeljuje se procesu, 3. proces pristupa resursu, 4. proces oslobadja resurs.
Ako je resurs zauzet, proces se suspendiše i kada se resurs oslobodi, proces se prevodi u aktivno stanje. Proces traži resusr preko sistemskog poziva: OPEN, REQUEST.

Četiri uslova za medjusobno blokiranje-Grupa procesa je u medjusobnom blokiranju ako svaki proces čena na neki događaj koji može da generiše samo neki drugi proces. Događaj je obično oslobađanje nekog resursa. Uslovi: medjusobno isključivanje; svaki resurs je ili slobodan ili ekskluzivno dodeljen jednom procesu, proces kome je dodeljen jedan resurs može da zahteva drugi resurs, nema "nasilnog" oslobadjanja resursa; samo proces kome je dodeljen resurs može da ga oslobodi, mora da postoji čekanje u krug, tj. dva ili više procesa čekaju na resurse koje drže drugi procesi.