LEKCIJU KONSPEKTI

# OPERĒTĀJSISTĒMAS

RTU 2003

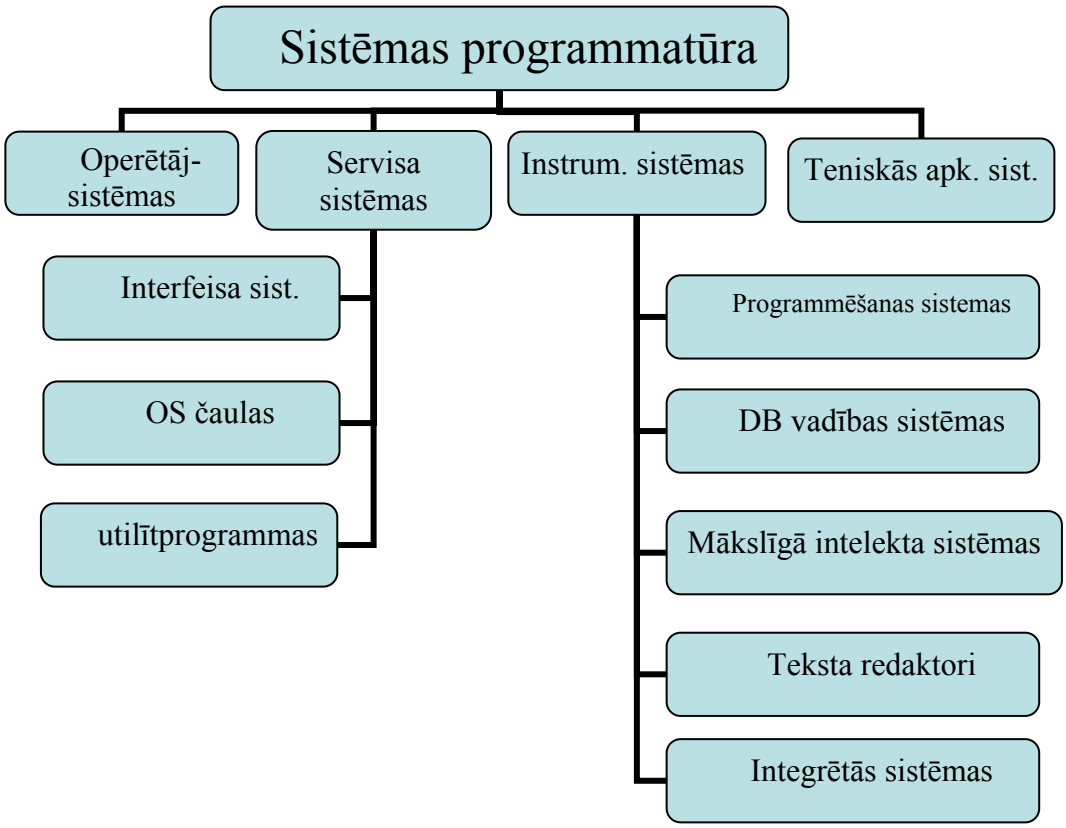
# Saturs

# 1. Lekcija

## Sistēmas programmatūras struktūra.

Datora programmatūra ir visa veida programmu kopums, kas nodrošina datu apstrādes sistēmas efektīvu darbību un lietotāja apkalpošanu. Pēc funkcionālas pazīmes datora programmatūru var nodalīt divās daļās:

- sistēmas programmatūra (izmanto programmas produktu izstrādāšanai, izpildīšanai, lietotāja apkalpošanai)
- lietišķā programmatūra, kuru izmanto konkrēta uzdevuma risināšanai.




### Operētājsistēmas

Organizē programmas izpildi un lietotāja apkalpošanu

### Servisa sistēmas

Paplašina OS iespējas

### Instrumentu sistēmas

paredzētas programmatūras izstrādāšanai.

### Apkalpojošās sistēmas

paredzētas datora aparatūras testēšanai un bojājumu meklēšanai.

# OS uzdevumi un funkcijas, lietotāja un programmas interfeiss

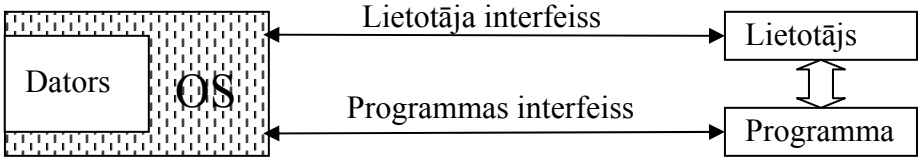
OS ir programmu komplekss, kas vada datu organizēšanu un programmu izpildīšanu datorā, nodrošina aparatūras un programmatūras kopdarbību, resursu racionālu izmantošanu un sadarbību ar lietotāju. OS galvenos uzdevumus var sadalīt šadās sasāvdaļās:

- ērtība
- efektivitāte
- attīstības iespēja

OS jābūt tādā organizācijā, kura pieļauj jaunu pielikumu un sistēmas funkciju efektīvu izstrādi, testēšanu un ieviešanu tādā veidā, lai netraucētu sistēmas normālai funkcionēšanai. Resursi ir datu apstrādes sistēmas līdzekļi, kas piedalās darba veikšanā. Resursu vadīšana nozīmē:

- pieejas pie resursiem atvieglošana- realizēšana ļauj paslēpt datora aparatūras īpatnības un iedot lietotāja rīcībā virtuālo mašīnu ar būtiski vieglāku vadīšanu.
- resursu sadalīšana starp konkurējošiem procesiem- raksturīga tādām operētājsistēmām, kas nodrošina dažu programmu vienlaicīgu izpildīšanu. OS uztur divus interfeisus, kuru līmenis ir augstāks par aparatūras interfeisu.

OS uztur 2 interfeisus, kuru līmenis ir augstāks par aparatūras interfeisu:



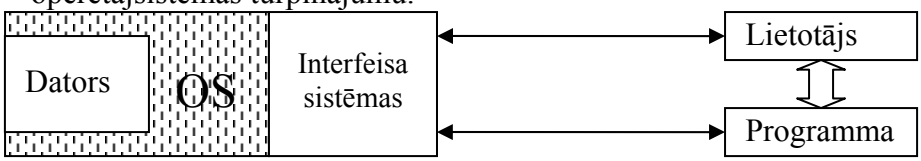

1.zīmeejums

- Lietotāja interfeiss- saskarne, kura nodrošina informācijas apmaiņu starp lietotāju un datu apstrādes sistēmas
- Programmas interfeiss – pakalpojumu komplekss, kas atbrīvo programmētājus no rutīnas operāciju kodēšanas

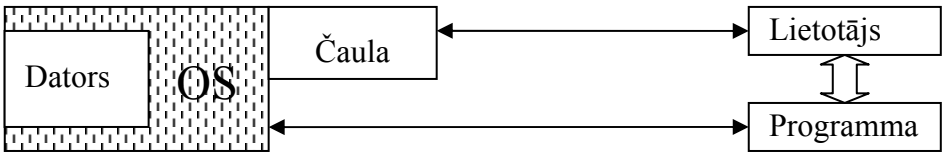
## Serviss sistēmas

Servisa sistēmas ir tādas sistēmas, kas paplašina un papildina OS lietotāju un programmu interfeisus

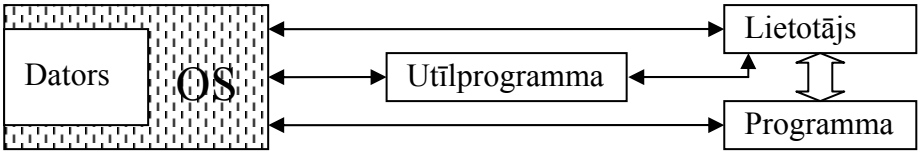
- Interfeisa sistēmas, kuras modificē lietotāja interfeisu un dažkārt arī realizē resursu sadalīšanas funkcijas). Bieži šādas interfeisa sistēmas uzskata par operētājsistēmas turpinājumu.



- Interfeisa sistēmas, kuras modificē tikai lietotāju interfeisu (shell- čaulas). [Norton Commander, VC, FAR manager]




- Utīlprogrammas - apkalpojoša veida programmas, kas ļauj bagātināt lietotāja interfeisu.




## Hierarhiskais OS modelis.

Lai būtu kopējais priekšstats par OS ir vērts raksturot hierarhiskas OS modeli, kas sastāv no sekojošiem līmeņiem:

| Līmenis | Līmeņa nosaukums | Objekti | Operāciju piemēri |
|---|---|---|---|
| 13. | Čaula | Programmēšanas lietotāju čaula | Čaulas komandu valodas instrukcijas |
| 12. | Lietotāju procesi | Lietotāju procesi | Procesa pabeigšana, apturēšana, atjaunošana |
| 11. | Katalogi | Katalogi | Izveidošana, dzēšana, meklēšana |
| 10. | Iekārtas | Ārējas iekārtas (printeris, monitors, tastatūra) | Atvēršana, aizvēršana, nolasīšana, ierakstīšana |
| 9. | Failu sistēma | Faili | Izveidošana, dzēšana, atvēršana, aizvēršana, lasīšana, ierakstīšana |
| 8. | Komunikācijas | Konveijeri | Izveidošana, dzēšana, atvēršana, aizvēršana, lasīšana, ierakstīšana |
| 7. | Virtuālā atmiņa | Segmenti, lappuses | Lasīšana, ierakstīšana, ienese |
| 6. | Lokālā sekundārā atmiņa | Datu bloki, iekārtu kanāli | Lasīšana, ierakstīšana, iedalīšana, ienešana |
| 5. | Primitīvi procesi | Primitīvi procesi, semafori, procesu saraksti | Apturēšana, izpildīšanas atjaunošana, gaidīšana, signāla pārraide |
| | | | |
| 4. | Pārtraukumi | Pārtraukumu apstrādes programmas | Izsaukums, maskēšana, atkārtošana |
| 3. | Procedūras | Procedūras, izsaukumu steki | Izsaukums, atgriešana |
| 2. | Komandu kopa | Skaitļošanas steks, mikroprogrammu interpretators, dati | Ielāde, saglabāšana, saskaitīšana, atņemšana, zarošana |
| 1. | Elektroniskās shēmas | Reģistri, kopnes, atmiņas šūnas... | Notīrīšana, pārsūtīšana, aktivēšana |

1.-4.līmenis faktiski nepieder OS sastāvdaļai, bet pieder pie procesora aparatūras komponentiem, bet piemēram, 4.līmenī jau var pamanīt tādus OS elementus, kā pārtraukuma apstrādes programmas

5.līmenī tiek ieviests procesa jēdziens, zem kura saprot programmu izpildes gaitā. Lai nodrošinātu dažu procesu vienlaicīgu gaitu, OS jānodrošina iespējas apturēt un atjaunot procesus, kā arī sinhronizēt tos. Šajā līmenī parādās viena no vienkāršākajām signālu pārraides koncepcijām- semofors.

6.līmeņa komponenti sadarbojas ar datora palīgatmiņas ierīcēm. Šajā līmenī notiek magnētiskās galviņas pozicionēšana un datu bloku fiziska pārraide. Darba plānošanai un paziņošanai process par pieprasītās operācijas pabeigšanu šis līmenis izmanto 5-tā līmeņa komponentes.

7.līmenis veido procesu adrešu platību. Šī platība tiek organizēta bloku veidā. Blokus var pārvietot starp pamatatmiņu un palīgatmiņu. Ja vajadzīgais bloks neatrodas pamatatmiņā, tad šis līmenis pārraida pieprasījumu 6.līmenim par bloka pārraidi.

Līdz šim gāja runa par OS sadarbību ar procesoru, augstāko līmeņu komponenti sadarbojas jau ar ārējiem objektiem- t.i. perifērijas ierīces, citi datori.

8.līmenis ir atbildīgs par informācijas apmaiņu starp procesiem. Viens no jaudīgākajiem instrumentiem šajā jomā ir konveijeri. Konveijers ir loģiskais datu pārraides kanāls, kad viena procesa izejas dati ir cita procesa ieejas dati.

9.līmenis nodrošina failu ilgtermiņa glabāšanu. Šajā līmenī dati tiek uzskatīti kā abstrakti objekti ar mainīgu garumu.

10.līmenī tiek organizēta pieeja ārējām ierīcēm ar standarta interfeisiem.

11.līmenis uztur sakarus starp sistēmas resursu un objektu ārējiem un iekšējiem identifikatoriem. Ārējais identifikators ir vārds, ko izmanto lietojumprocess vai lietojums, iekšējais identifikators ir adrese vai cits identifikators, ko OS zemākie līmeņi izmanto objektu meklēšanai vai vadībai.

12.līmenis nodrošina procesu atbalstīšanas pilnfunkcionālos līdzekļus. Šajā līmenī procesora reģistru informācija tiek izmantota procesu noregulētai vadībai.

13.līmenis nodrošina OS sadarbību ar lietotāju. Šo līmeni sauc par čaulu un tā piedāvā lietotājam servisus. Čaula pieņem komandas, interpretē tās, rada vajadzīgos procesus un pārvalda tos. Šajā līmenī varētu būt realizēts grafiskais interfeiss, kas ļauj lietotājam izvēlēties komandas no izvēlnes un izpildes rezultātus attēlo uz ekrāna.

# 2. Lekcija

## OS funkcionālie komponenti

OS funkcijas var sagrupēt saskaņā ar lokālo resursu rīkiem, kurus vada OS saskaņā ar specifiskiem uzdevumiem, kurus var pielietot pie visiem resursiem. Dažkārt tādas OS funkcijas sauc par apakšsistēmām.

Svarīgākās resursu vadības apakšsistēmas:

- Procesu vadība
- Atmiņas vadība
- Failu un ārējo iekārtu vadība

Apakšsistēmas, kuras ir kopējas visiem resursiem:

- Lietotāja un programmas interfeisi
- Datu aizsardzība un administrēšana

### Procesu vadības apakšsistēma

Svarīgākā OS daļa, kas ietekmē datora funkcionēšanu. Katram jaunizveidotam procesam OS ģenerē sistēmas informatīvās struktūras, kas satur datus par resursiem, kuri vajadzīgi procesam un kuri faktiski iedalīti procesam. Šajā struktūrā saglabā arī datus par procesa atrašanās vēsturi sistēmā, par procesa pašreizējo stāvokli un prioritāti. Šos datus OS izmanto, kad pieņem lēmumu par resursu iedalīšanu procesam. Multiprogrammu sistēmās OS organizē rindas pie resursiem, aizsargā resursus, nodrošina resursu koplietošanu, pārslēdz procesu stāvokļus, sinhronizē procesu izpildi un sadarbību.

### Atmiņas vadības apakšsistēmas

Nodrošina fiziskās atmiņas sadalīšanu starp visiem dotajā momentā eksistējošiem procesiem, procesa koda un datu ielādi atmiņas apgabalos un to aizsardzību. Viens no populārākajiem (un galvenajiem) atmiņas vadības paņēmieniem ir virtuālās atmiņas paņēmiens.

### Failu un ārējo iekārtu vadības apakšsistēmas

OS iespēja izolēt pielietojumus no reālas aparatūras īpatnībām tiek realizēt failu sistēmas jēdzienā. Dati, kas tiek glabāti ārējā atmiņā ir organizēti failos, kuriem ir simboliski vārdi. Failu sistēmas izpilda simbolisko vārdu pārveidošanu fiziskajās adresēs, izpilda visas faila operācijas, organizē pieeju failiem un aizsargā tos no nesankcionētas pieejas. Failu sistēma cieši sadarbojas ar ārējo iekārtu vadības apakšsistēmu, kas pēc failu sistēmas vaicājumiem nodrošina datu pārraidi starp magnētiskajiem diskiem un operatīvo atmiņu. Šī apakšsistēma organizē interfeisus visām ieslēgtām iekārtām un uzturēt tādu interfeisu ir viena no svarīgākajām un grūtākajām OS funkcijām (draiveri).

### Datu aizsardzība un administrēšana

OS sastāvā jābūt līdzekļiem, kas nodrošina sistēmas drošību aparatūras un programmatūras darbības traucējumu gadījumos un aizsargā sistēmas komponentus no nesankcionētas pieejas. Piemēram- ieejas loģiskā kontrole (vārds un parole), pieejas pie resursiem ierobežošana, audits, iekārtu rezervēšana. Visas aizsardzības funkcijas ir cieši saistītas ar administrēšanu

## Lietišķās programmēšanas interfeiss[1]

API ļauj programmētājam izmantot OS iespējas. No lietotāja API funkcijas ir paslēptas ar lietotāja čaulas palīdzību. Lietojumu izstrādātājiem konkrētās OS īpašības tiek attēlotas API īpašību veidā. API standartizēšana ļauj izstrādāt pārnesamus pielietojumus, kurus var izmantot atšķirīgās OS vidēs. Lai izmantotu API funkcijas, pielietojumi izpilda sistēmas vaicājumus, kuru realizācija ir atkarīga no OS struktūras organizācijas un aparatūras īpatnībām.

## Lietotāja interfeiss

Lietotāju interfeisus OS var uzturēt divos veidos.

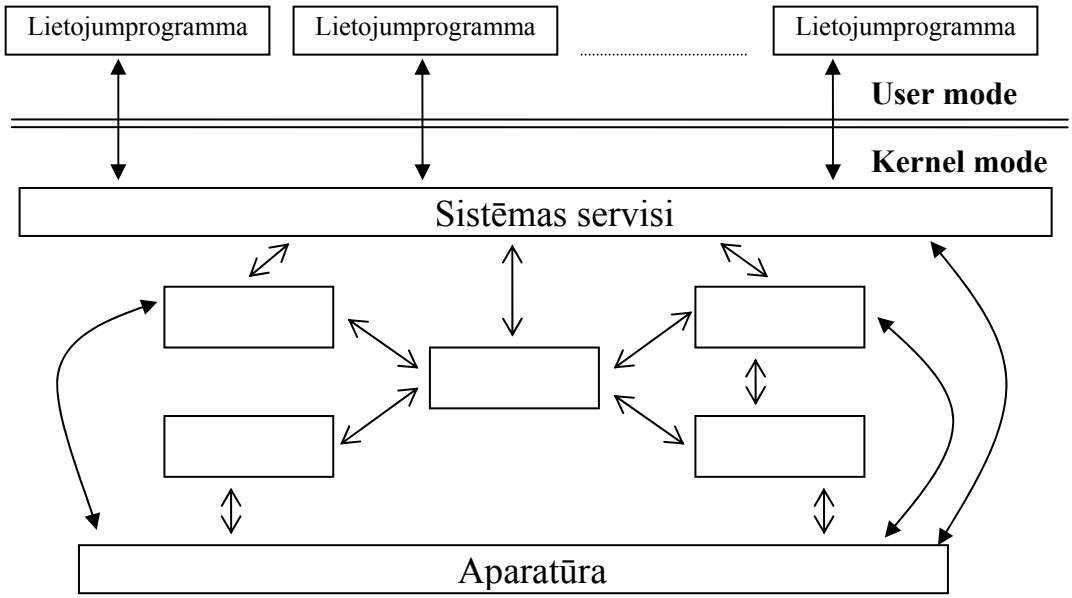
- Burt-ciparu
- Grafisko interfeisu veidā

Burt ciparu interfeisā, lietotāja rīcībā ir komandu valoda, kas atspoguļo sistēmas funkcionālās īpatnības. Komandas var izpildīt vai nu interaktīvajā režīmā vai komandu faila veidā. Komandu ievadi var būtiski atvieglot, ja OS uztur lietotāja grafisko interfeisu.

## OS struktūru organizācija

Eksistē vairāki OS kodola strukturēšanas veidi

### Monolīta OS

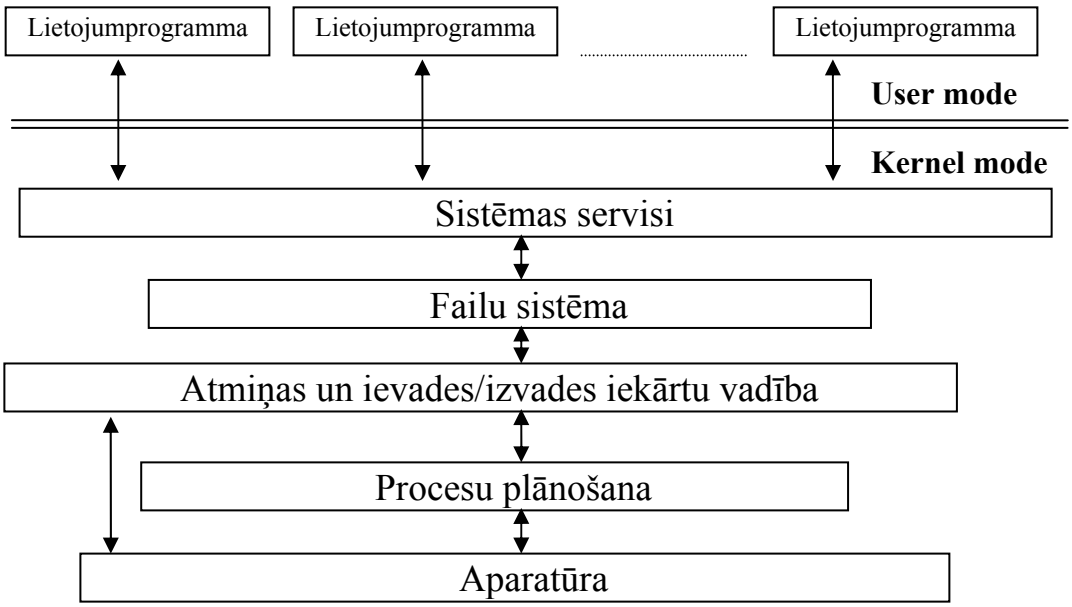



Nelielas OS struktūra, bieži organizēta kā procedūru kopa, katru no kurām var izsaukt jebkurā lietotāju procedūrā. Tāda struktūra nodrošina datu izolēšanu. Dažādos koda gabalos tiek izmantota informācija par visas sistēmas ierīkošanu. Tādu OS paplašināšana ir ļoti sarežģītas process, jo vienas procedūras mainīšana var izsaukt kļūdas citās OS daļās, kuras uz pirmo skatienu nav saistītas ar šo procedūru. Visi OS pielikumi ir atdalīti no pašas OS. OS kods ir izpildīts priviliģētā režīmā (kernel mode) un tam ir pieeja sistēmas resursiem un aparatūrai. Pielikumi vai lietojumprogrammas izpildās lietotāju režīmā, kurā ir ierobežota pieeja sistēmas datiem. Kad lietojumprogramma izsauc sistēmas servisu, procesors pārķer izsaukumu un pārslēdz

[1] Application Programming Interface- API

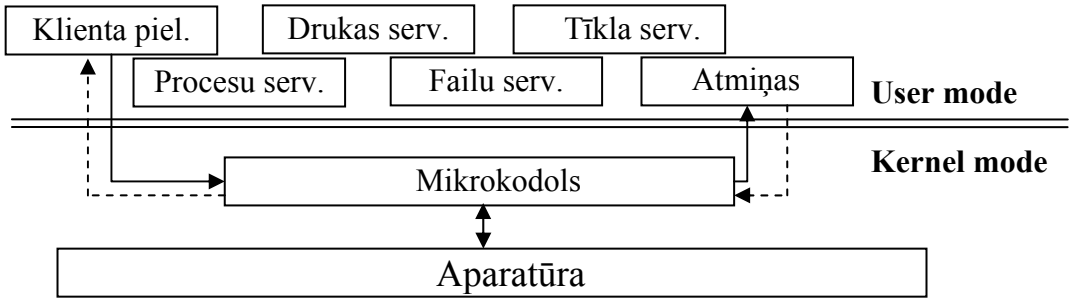
plūsmu kodola režīmā. Faktiski notiek režīma pārslēgšana. Kad sistēmas izsaukums tiek izpildīts, OS pārslēdz plūsmu atpakaļ lietotāju režīmā un dod iespēju programmai turpināt izpildīšanu.

### Slāņu OS




Paredz OS sadalīšanu uz moduļiem, kuri tiek izvietoti dažādos slāņos. Katrs modulis piedāvā funkcijas, kuras var izsaukt citi moduļi, bet modulis var izmantot kodus tikai no zemāk izvietotiem slāņiem. Šādas OS priekšrocības- katra slāņa kods var saņemt pieeju tikai interfeisam un datiem no zemāk izvietotiem slāņiem, kas samazina koda ar neierobežotu varu izmēru (apjomu). Šī struktūra ļauj izstrādes un skaņošanas gaitā uzsākt darbu no zemāka slāņa un pakāpeniski pievienot citus slāņus. Slāņu organizācija atvieglo OS paplašināšanas iespējas. [viena no šīm- OS Multics]

### Klient-servera OS modelis



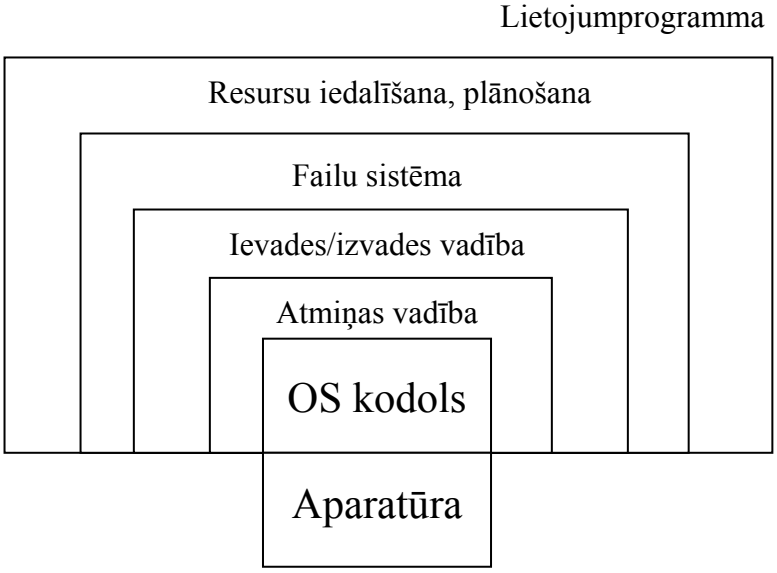

Šajā gadījumā OS tiek sadalīta uz procesiem, katrs no kuriem realizē vienu servisa kopu, piemēram, atmiņas sadalījums, procesu vadība, procesu plānošana utt. Katrs process izpilda vienu funkciju.

Katrs OS serviss funkcionē lietotāju režīmā un pārbauda vai kāds klients nav pieprasījis apkalpošanu. Klients, par kuru var uzskatīt vai nu lietojumprogrammu vai jebkuru sistēmas servisu, pieprasa servisa izpildīšanu sūtot serverim paziņojumu. OS kodols (mikrokodols) pārsūta paziņojumu serverim, kurš izpilda pieprasītās darbības. Serveris izpilda apkalpošanu un mikrokodols pārsūta klientam rezultātus cita

paziņojuma veidā. Šajā gadījumā OS sastāv no neliela izmēra autonomiem komponentiem.

Par cik visi servisi tiek izpildīti kā lietotāja režīma atsevišķi procesi, tad avārija ar kādu no tiem neietekmē citu OS daļu darbību. Dažādi servisi var izpildīties uz dažādiem procesoriem vai datoriem, kas dod iespēju izmantot šo pieeju dalītās OS.

## Objektu modelis




*OS hipotētisks modelis*

Saskaņā ar objektorientēto metodoloģiju, no sākuma vajag izdalīt datus, ar kuriem strādās programma. OS par tādiem datiem uzskata sistēmas resursus (failus, atmiņas blokus utt.). Kad izstrādā sistēmu, kuras orientētas uz datoriem, tad izstrādes galvenais mērķis ir tādas programmatūras izveidošana, kuru var viegli un lēti izmainīt un pielāgot citai videi. Viens no veidiem kā var minimizēt izmaiņas objektorientētās programmās ir paslēpt datu fizisko attēlošanu objektu iekšienē. Objekts ir datu struktūra, kuras fiziskais formāts ir paslēpts tipa definēšanā.

Katru sistēmas resursu, kuru kopīgi var izmantot vairāki procesi, var realizēt kā objektu un apstrādāt ar objektu servisiem.

Priekšrocības:

- Samazināta izmaiņu ietekme
- Unificēta pieeja pie OS resursiem un darbs ar tiem
- Vienkāršota objektu aizsardzība

[Windows NT, Windows 2k, Windows XP, ...]

# 3. Lekcija

## MS-DOS attīstības etapi, priekšrocības un trūkumi

Pirmā pārdošanas versija – 1981.g
MSDOS 1 – 1.10
MSDOS 2 – 2.10 (HDD atbalsts)
MSDOS 3 – 3.30 (IBM AT support)
MSDOS 4 – 4.01
MSDOS 5.0
MSDOS 6 – 6.22 (<- Pēdējā atsevišķā MSDOS versija)
MSDOS bija par pamatu 1 OS saimei – Windows 9x.
Mūsdienās iekš Windows ir iespējams emulēt DOS darbu.
Priekšrocības:

- ērts un vienkāršs lietotāja interfeiss
- iespēja izstrādāt .bat failus
- hierarhiska failu struktūra
- iespēja izmantot failu šablonus darbībās ar failiem
- iespēja realizēt secīgu un tiešu pieeju faila saturam
- konvejeri un I/O pārsūtīšana komandu valodas līmenī
- sistēmu struktūras modularitāte
- operatīvās un ārējās atmiņas neliels apjoms, kas nepieciešams sistēmas failu glabāšanai
- iespēja izveidot virtuālus diskus, *kas paātrina informācijas apmaiņu*
- Trūkumi:
- nav līdzekļu, kas aizsargātu pieeju pie resursiem
- pieejamās RAM ierobežojums 640 Kb

## Struktūra un pamatkomponentu funkcijas

1) Sistēmas ielādētājs (SB – System Booter)
2) BIOS paplašināšanas modulis (EM – Extension Module)
3) Ierīču ārējie draiveri
4) DOS bāzes modulis (BM – Basic Module)
5) Komandu interpretātors (CI- Command Interpreter)
6) DOS utilītprogrammas
7) MS-DOS Shell čaula (doss*.*)
8) instrumentālie līdzekļi – BASIC (progr. val.)
   + BIOS (Basic Input/Output System)
   + Non-system bootstrap (NSB- Non-System Bootstrap, ievieto ar fdisc /mbr)
   Funkcijas

- DOS ielādēšanas gaitā
- DOS funkcionēšanas gaitā

Funkcijas:

| ***DOS komponente*** | ***Atrašanās vieta*** | ***ielādes gaitā*** | ***funkc. gaitā*** |
|---|---|---|---|
| BIOS | pastāvīgā atmiņā | 1. Datora ierīču testēšana<br>2. Apakšējā līmeņa Int vektoru inicializācija<br>3. NSB nolasīšana atmiņā | 1. Standarta perifērijas ierīču vadīšana |
| NSB | MBR (Cietā diska pirmais sektors) | Ielādē un palaiž SB | N/A |
| SB | Katra loģiskā diska sākuma sektorā | 1. EM BIOS un BM DOS nolasīšana atmiņā<br>2. EM BIOS palaišana | N/A |
| EM BIOS | IO.SYS | 1. Aparatūras stāvokļa pārbaude un *iekšējo* perifērijas ierīču inicializācija<br>2. Saspiestā diska draivera DBLSPACE.BIN pieslēgšana<br>3. DOS konfigurēšana atbilstoši Config.Sys<br>4. Apakšējā līmeņa pārtraukumu vektoru inicializācija un pārkārtošana<br>5. BM DOS palaišana | BIOS iespēju paplašināšana |
| Ierīču draiveri | atsevišķi faili | N/A | 1. nestandarta perifērijas iekāru vadīšana<br>2. standarta perifērijas iekārtu nestandart vadīšana |
| BM DOS | Msdos.Sys | 1. DOS iekšējo tabulu inicializēšana<br>2. Augšējā līmeņa pārtraukumu inicializācija<br>3. CI nolasīšana atmiņā un palaišana | 1. Datora resursu un izpildāmo programmu vadīšana |

| *DOS komponente* | *Atrašanās vieta* | *ielādes gaitā* | *funkc. gaitā* |
|---|---|---|---|
| CI | Command.com | 1. 3 apstrādājamo vektoru (20h, 21h, 22h) inicializācija<br>2. AUTOEXEC.BAT | 1. DOS komandu pieņemšana no tastatūras<br>2. Iekšējo komandu izpildīšana<br>3. Komandu failu izpildīšana<br>4. Programmu ielāde atmiņā<br>5. Pārtraukumu apstrāde pēc uzdevuma pabeigšanas |
| Utility prog. | atsevišķi faili | | 1. Ārējo DOS komandu izpildīšana<br>2. Servisa pakalpojumu realizēšana interaktīvajā režīmā |
| MS DOS Shell | doss*.* | | Lietotāja interfeisa līmeņa paaugstināšana |
| Instrumentāl ie līdzekļi | atsevišķi faili | | Izmanto programmu izvadei un teksta redaktēšanas funkciju veikšanai |

## Sistēmas diska struktūra

Sistēmas disks ir tāds disks, no kura ielādējas sistēma.
HDD vai FDD

- Starta sektors
- IO.SYS
- MSDOS.SYS
- COMMAND.COM
- CONFIG.SYS
- AUTOEXEC.BAT

## MSDOS Komandu klasifikācija

DOS komandas nodrošina lietotāja sadarbību ar sistēmu.
Pēc funkcionālās pazīmes un lietošanas tās var sadalīt 5 grupās:

1. Vispārējās
    1. Disku manipulācijas komandas (fdisk, chkdsk)
    2. Katalogu manipulācijas komandas (cd, md, tree, xcopy)
    3. Failu manipulācijas komandas (move, copy, attrib, del, ren, type)

4. Simbolu iekārtu vadība (cls, print, type)
5. Sistēmas rekonfigurēšanas komandas (date, path, set, time)
6. Sistēmas vadīšanas komandas (command, exit)
7. Informatīvās komadas (attri, date, dir, time, tree)

2. Instrumentālās (debug, edit)
3. Filtri (find, sort, more)
4. Komandu un failu komandas (tikai *.bat failos) (echo, for, goto, if, choice)
5. Sistēmas konfigurēšanas komandas (CONFIG.Sys)

   Break=
   Device=
   devicehigh=
   buffers=
   dos=
   stacks=

## Datora pārtraukumu sistēma

DOS funkcionēšanas galvenais mehānisms ir pārtraukumu sistēma. Par pārtraukumu sauc situāciju, kas prasa kādu procesora darbību dažādu notikumu rašanās rezultātā. Pārtraukumu apstrāde – standarta darbību kopums, ko OS pārtraukuma gadījumā veic ar programmu un aparatūras palīdzību. Pārtraukumu apstrāde sastāv no pārtraukuma tipa analīzes un attiecīgās pārtraukuma apdarinātāja darbības.

Pārtraukumu apdare – speciāla I/O rutīna, kas veic pārtraukuma apstrādi (apkalpošanu)

IBM PC ir šādas pārtraukumu kategorijas:

- Ārējā aparatūras pārtraukumus izraisa ārpus procesora notikumi. Šādu pārtraukuma signālus izsauc perifērijas iekārtas. Signāli parādās sistēmas maģistrālē un tiek apstrādāti ar pārtraukuma kontrolieri. Piemēram, taustiņa nospiešana.
- Iekšējās aparatūras pārtraukumus izsauc pats mikroprocesors (dalīšana ar nulli, ...)
- Programmas pārtraukumi.

IBM datori uztur 256 pārtraukumus ar kodiem no 0 līdz 255 un prioritāti no 0 līdz 15.

Lai apstrādātu pārtraukumus, RAM sākumā ievieto pārtraukumu vektoru tabulu (1024 baiti).

Katrs tabulas lauks ir pārtraukuma vektors, kas satur atmiņas adresi, kur atrodas rutīna, kas apkalpo šo pārtraukumu.

Pārtraukuma apstrāde sastāv no šādiem soļiem:

1. Pārtraukuma koda formēšana un tā pārraide mikroprocesorā
2. Mikroprocesora tekošā stāvokļa saglabāšana stekā
3. Izsauc attiecīgo rutīnu no pārtraukumu tabulas
4. Izpildās pārtraukuma apstrādes programma
5. Atjaunojas CPU stāvoklis

IO.SYS apstrādā pārtraukumus no 00h līdz 20h, bet MSDOS.Sys – no 23h līdz 59h.

# 4. Lekcija

## DOS failu sistēma

DOS failu sistēma ir OS daļa, kas iekļauj:

- visu failu kopumu uz magnētiskiem diskiem
- datu struktūru komplektu, ko izmanto failu vadībai
- sistēmas programmu līdzekļu kompleksu, kas ļauj realizēt failu organizāciju un failu operācijas

DOS failu sistēmas galvenās funkcijas ir sekojošas:

- noteikt failu organizācijas iespējamos veidus
- realizēt pieejas metodes
- noteikt failu struktūras organizācijas veidus
- nodrošināt iespējas izmantot failu struktūras manipulēšanas līdzekļus

Failu organizācijas veids ir failu loģiskā struktūra, kas raksturo faila komponentes un tās savstarpējās saites. Organizācijas veidi var būt:

- tiešais
- secīgais
- indeksētais

Praksē failu organizācijas veidi tiek noteikti ar pieejas metodēm, kuras ļauj izmantot sistēmā. Pieejas metode ļauj atrast ierakstu failā, lai nodrošinātu to apstrādi un modificēšanu. Pieejas metodes realizē sistēmas programmas. Failu struktūra ir failu kopums un to savstarpējās saites. Faktiski OS tiek raksturotas ar hierarhiskām failu struktūrām. DOS failu sistēma pēc struktūras ir sadalīta starp BM un BIOS. AR failu struktūru DOS atšķir 2 veidu ierīces:

- simbolu ierīces
- bloku ierīces

Informācijas apmaiņa starp operatīvo atmiņu un simbolu ierīci notiek secīgi viens baits pēc otra. Pie šīs grupas pieder ievad/izvad ierīces u.c perifērijas ierīces. DOS rezervē šo iekārtu vārdus:

- LPT1 vai PRN – paralēlā interfeisa 1.adapteris (parasti printeris). Paralēlais interfeiss ir saskarne, kas nodrošina visu baitā esošo bitu vienlaicīgu pārraidi. Adapteris ir ierīce, kas nodrošina atšķirīgu iekārtu vai sistēmu sadarbību.
- LPT2 un LPT3 – 2. un 3. adapteris
- COM1 un AUX – seriāla interfeisa 1. adapteris (visbiežāk modems). Seriālais interfeiss ir saskarne, kas nodrošina secīgu datu pārraidi pa vienam bitam. Modems ir funkcionāla ierīce, kas ciparu signālu pārveido analogu signālā un otrādi.
- COM2, COM3 un COM4 – 2.,3.,4

NUL – fiktīva ierīce, kuru var izmantot programmas atkļūdošanas gadījumā

CON – konsole. Parasti sastāv no tastatūras un monitora

CLOCK – reāla laika pulkstenis.

Katrai ierīcei ir atbilstošs draiveris – OS programma, kas apkalpo ierīci. Informācijas apmaiņa starp operatīvo atmiņu un bloku ierīci notiek pa blokiem 512 baitu apjomā. Izmanto alfabēta burtus. MS DOS vienmēr ierīce ir aktīva, ir aktīvais disks un šī diska vārdu nevajag uzrādīt komandās

DOS failu sistēma atbalsta 2 funkcionāli dažādus failu tipus. Tas ir:

- vienkārši faili

- katalogi

Vienkāršie faili satur brīva rakstura informāciju un DOS uzskata to kā loģisko ierakstu secību. DOS atšķir 2 formātu failus:

- bināri faili – izpild programmu kodi, atmiņas att.
- Teksta faili

Ar katru failu ir saistīti sekojoši parametri:

- Vārds
- Atribūti
- Veidošanas laiks
- Veidošanas datums
- Faila izmērs

Faila atribūti raksturo faila izmantošanas veidus un pieejas tiesības. MS DOS ļauj uzrādīt šādus atribūtus:

R – read only

A – archive (arhivējams fails – failam, pēc pēdējām izmaiņām tajā, nav rezerves kopijas). Šo atribūtu izmanto kopēšanas komandas.

H – hidden (slēptais fails)

S – system (sistēmas fails) Nozīmē ka fails saistīts ar noteikto vietu uz diska. DOS failu sistēma nodrošina dažādas failu operācijas:

- Veidošana
- Dzēšana

} Atmiņas iedalīšana vai atņemšana

- Magnētiskās galviņas pozicionēšana
- Pārsūtīšana
- Pārdēvēšana
- Lasīšana
- Rakstīšana

Izpildot failu organizācijas failu sistēma identificē pieprasījumu izvēlas draiveri. Draiveris nodrošina ierīces adaptera vadīšanu. Pēc operācijas pabeigšanas draiveris atdod sava darba rezultātus kodolam un šie rezultāti tiek pārsūtīti programmai, kas pieprasa operāciju. Ievad/izvades operācijas failu sistēmā realizē tikai taktēšanas veidā, tas nozīmē, ka programmas izpildīšana tiek apturēta līdz failu operācijas beigām.

Katalogi satur sistēmas izziņas informāciju par failu koku, kas tiek apvienoti pēc kaut kādas neformālas pazīmes. Katalogi nodrošina atbilstību starp failu vārdiem un failu raksturojumu, kurus OS izmanto failu vadībai.

Magnētiskā diska formatēšanas gaitā OS vienmēr veido saknes katalogu, kas sastāv no 32 baitu datiem- ierakstiem, kuros tiek glabāta informācija par failiem, kas atrodas uz diska. Kataloga ierakstu struktūra ir sekojoša:

1. faila vārds 8 baiti
2. paplašinājums 3 baiti
3. faila atribūtu binārais kods 1 baits
4. rezerves lauks 10 baiti
5. izveidošanas laiks 2 baiti
6. izveidošanas datums 2 baiti
7. pirmā klastera, ko aizņem fails, numurs 2 baiti
8. faila izmērs 4 baiti

Atribūtu kodi raksturo faila statusu, šajā laukā tiek saglabāti 6 rādītāji, katrs no kuriem var pieņemt vērtību 0 vai 1 un aizņem bitu

0.bits – atribūts R

1.bits – atribūts H

2.bits – atribūts S
3.bits – atribūts sējuma iezīme (volume)
4.bits – pazīme, ka fails ir katalogs
5. baits – atribūts A

Saknes katalogam vieta uz diska ir fiksēta, t.i., sākot no sektoriem pēc FAT tabulas. Apakškatalogi tiek glabāti kā vienkārši faili diska datu apgabalā. Apakškatalogi atšķiras no parasta faila ar to ka faila atribūtu 4 bits atrodas stāvoklī 1 un faila izmērs ir 0. Failu sistēmas fiziskā organizēšana nosaka principus pēc kuriem faili, katalogi un sistēmas informācija tiek ievietoti uz reālas ierīces. Pašlaik par tām ierīcēm izmanto diskus.

Griešanas shēma ir sekojoša:

Lietotājs griežas pie faila uzrādot faila vārdu, OS griežas pie kataloga, atrod ierakstu, kas atbilst failam, nosaka sektorus, kuros fails ir izvietots, tikai pēc tam OS izpilda vajadzīgās darbības, izpildot pieeju datiem. Sistēma izmanto speciālo tabulu (FAT tabulu). FAT ir svarīgākā diska failu struktūras daļas, kas realizē diska platības iedalīšanu failiem un nodrošina pieeju tiem. Diska loģiskā nodaļa, kura ir formatēta FAT failu sistēmai, iekļauj sekojošus apgabalus:

- darba sektors (kopija, kas satur informāciju par failu un katalogu izvietošanu uz diska) Ir rezerves kopija FAT2
- saknes katalogs (root directory) sastāv no ieraksta – 32 baitiem. 1 ieraksts raksturo katru failu
- datu apgabals – paredzēts visu failu un paredzēts visu failu apakškatalogu izvietošanai

Atmiņu iedala no diska datu apgabalā un par minimālo diska apgabalu izmanto klasteru. FAT tabula sastāv no rādītāju masīvā, kuros skaits sakrīt ar klasteru skaitu diska datu apgabalā. Rādītājā varētu būt šādas vērtības, kas raksturo ar to saistīto klastera stāvokli:

1) klasteris ir brīvs – kods FAT tabulā ir 3 nulles
2) klasteris ir ar defektu FF7
3) klasteru izmanto failam un šis klasteris nav faila pēdējais klasteris. Rādītājs satur nākamā klastera numuru.
4) klasteris ir pēdējais failā FFF.

Failam, kuru vajag saglabāt uz diska, izpilda vajadzīgo (noteikto, veselo) klasteru skaitu. Izdalītie klasteri var atrasties diskā dažādās vietās. Tādus failus sauc par fragmentētiem.

Failam, kuru vajag saglabāt uz diska, iedala vajadzīgo klasteru skaitu

FAT 12 izmanto rādītājus ar 12 pozīcijām, kas ļauj atbalstīt līdz 4096 klasteriem datu apgabalā.

FAT 16 izmanto rādītājus ar 16 pozīcijām, kas ļauj atbalstīt līdz 65536 klasteriem datu apgabalā.

FAT 32 izmanto rādītājus ar 32 pozīcijām, ar ko var uzrādīt vairāk nekā 4 mlj klasteru.

Failu dzēšana no FAT failu sistēmas- katalogā atbilstošā ieraksta pirmajā baitā ievieto speciālo pazīmi par to, ka ieraksts ir brīvs un visos faila rādītājos ieraksta pazīmi- klasteris ir brīvs; pārējie ieraksta dati paliek bez izmaiņām, kas ļauj atjaunot failu, kas bija nodzēsts nejauši.

# 5. Lekcija

## Komandu un programmu izpildīšana dos vidē

Pieprasījumi komandu izpildīšanai tiek ievadīti no tastatūras, un pēc *ENTER* nospiešanas komandu sāk komandu interpretators. CI darbības ir atkarīgas no tā, kas tiek izvadīts: iekšējā, ārējā komandā vai komandu failu vārds.

Ja ir ievadīta iekšējā komanda, tad CI tranzītu modulis vienkārši izpilda to. Ja ievadīto pieprasījumu nevar identificēt kā iekšējo komandu, tad CI tranzītu modulis meklē failu struktūrā failu ar uzrādīto vārdu. Ja tāda faila nav, uz ekrāna tiek izvadīts paziņojums par kļūdu un CI atkal gaida ievadi. Ja ir atrasts .com vai .exe fails darbu sāk CI tranzītu modulis, kas izpilda sekojošas darbības:

1. Izveido DOS vides dublikātu(DOS vides kopija). DOS vide ir CI speciālais apgabals, kurā tiek saglabāti globālo mainīgo vārdi un to vērtības, ko izmato programmas, lai saņemtu noteiktu informāciju. Kad programmas tiek ielādēta izpildīšanai DOS vide tiek dublēta, kad process beidzas dublikāts tiek iznīcināts un oriģināls tiek aktivizēts.
2. Novieto un saglabā izpildāmā faila pilnu specifikāciju pēc DOS vides dublikāta.
3. Iedala izpildāmai programmai apgabalu RAM (operatīvajā atmiņā).
4. Iedalītās atmiņas apgabala sākumā tiek rezervēta atmiņa programmas segmenta prefiksam (PSP), ko izmanto, lai saglabātu nepieciešamo informāciju programmas izpildei.
5. PSP laukus aizpilda ar datiem:
   5.1. atmiņas izmērs, ko var izmatot programma
   5.2. DOS vides dublikāta adrese
   5.3. Argumenti no komandu rindas simbolu veidā
   5.4. tekošie pārtraukumu vektori 22h-24h, lai pēc programmas pabeigšanas būtu iespēja tās atjaunot un atgriezt DOS vidē.
6. Ielādē programmu atmiņā uzreiz pēc PSP.
7. Ja vajag (.exe fails) tiek izpildīta programmas pārvietošana un sākas programmas izpildīšana. Ja uz izpildīšanu ir ievadīts .bat fails, tad CI tranzītu modulis nolasa 1.rindu, analizē to un organizē atbilstošas komandas vai arī pašas programmas izpildi. Pēc tam nolasa 2.faila rindu, un interpretēšanas process atkārtojas.

## Atmiņas vadība dos vidē

Atmiņas vadībā ir pieejas nodrošināšana, atmiņas uzskaite un iedalīšana programmām. Adresējamā atmiņa- PC atmiņa, kuru var izmantot centrālais procesors. IBM PC datoriem atmiņas loģiski var sadalīt 3.daļās:

1) standarta atmiņas apgabals CMA- Conventional Memory Area (no 0-640Kb)
2) Augšējās atmiņas apgabals UMA- Upper Memory Area Diapazons (640Kb-1Mb)
3) Paplašinātās atmiņas apgabals XMA- Extended Memory Area(1Mb līdz UB-1, kur UB- kopējais atmiņas šūnu skaits mikroprocesorā).

MS DOS bez papilddraiveriem var adresēt atmiņu līdz vienam 1Mb. Tas ir mikroprocesora 8088/86 trūkums. Sākot no mikroprocesora 8026 var adresēt atmiņu jau virs viena 1Mb, bet strādājot DOS vidē ir piespiesti izmantot atmiņu līdz 1Mb. Tātad, faktiski anulē šo pirmo mikroprocesoru. Aizsargātais režīms- režīms, kurā var izmantot visu atmiņu. OS, kas izmanto aizsargāto režīmu var izmantot paplašināto atmiņu, tāpat kā standarta. Visu atmiņu, kura nav standarta sauc par papildatmiņu.

## Standartatmiņa

Standartatmiņu DOS var izmantot bez ierobežojumiem (izpildāmo komandu glabāšana, datu glabāšana, pati DOS izvietojas šajā atmiņā). Standartatmiņa tiek iedalīta pa blokiem. Katrā atmiņas blokā pirmie 16 baiti tiek iedalīti atmiņas vadības blokā (MCB), kas apraksta bloka apjomu un īpašnieku, šis bloks nodrošina arī saiti ar nākamo atmiņas bloku. Visi atmiņas bloki ir saistīti vienā virknē. Standarta atmiņas vadību pilnīgi izpilda MS-DOS.

## Attēlojamā atmiņa (Expended Memory Specification)

EMS koncepciju piedāvāja un kopīgi realizēja 3 firmas: Intel, MS un Lotus(?). Programmatūra un aparatūras līdzekļi, kas ir savietojami ar EMS, nodrošina papildatmiņu datu glabāšanai, lai programmas varētu to izmantot ar vienkāršo adresēšanu. Glabāt izpildāmās programmas šajā atmiņā nevar. EMS darbības princips tiek balstīts uz lpp aizvietošanas paņēmienu. UMA apgabalā tiek iedalīts 64K liels logs, kurā var attēlot 4 lpp no papildatmiņas. Ar šādu paņēmienu mikroprocesors tiek maldināts, jo izmantojot virtuālo adresēšanu tas griežas pie informācijas EMS logā, kaut arī konkrētās informācijas fiziskās adreses nepieder mikroprocesora adrešu platībai. Attēlojamo atmiņu anulēšanai izmanto draiveri EMM 3386.exe

## Paplašinātā atmiņa (eXtended Memory Specification)

XMS specifikāciju izstrādāja 1988.g.- MS, Inetl, Lotus, AST. Programmu draiveris, kas realizē XMS specifikāciju ir iekļauts DOS komplektā sākot no 4. versijas un ļauj pārsūtīt datus no standartamiņas uz paplašinātu, un otrādi. Izpildāmās programmas ievietot paplašinātajā atmiņā nav paredzēts. Tehniski pieeja paplašinātai atmiņai tiek realizēta ar mikroprocesora pārslēgšanu no reāla uz aizsargātu režīmu. Paplašinātas atmiņas izdalīšana notiek pa blokiem (EMB) un pēc speciāla pieprasījuma no programmas. Programma saņem savā rīcībā atdarinātāju un izmanto to, lai realizētu pieeju atmiņas blokiem.

## Augstā atmiņa (High Memory Area)

Mikroprocesors 8088/86 izmantoja 20 pozīciju adrešu maģistrāli, un nebija pieejams adresēt UMA pēdējo segmentu, jo adreses vecākā pozīcija- 21h tiek atmesta. Lai novērstu šo kļūdu mikroprocesora konstrukcijā bija iekļauts speciāls mezgls, kas pārvalda adrešu maģistrāles h20 stāvokli un aizliedz adresēšanu XMA pirmajam segmentam, bet tas mezglu var bloķēt un ar to parādās iespēja pieejai XMA1. Segmentam ar vienkāršu adresēšanu draiveris HIMEM.SYS realizē šo iespēju un atmiņas diapazons no 1M-1M+64K-16 tiek nosaukts par augsto atmiņu. Šajā atmiņā var izvietot gan datus, gan izpildāmās programmas. Pārsvarā šajā atmiņā ielādē DOS rezidentu moduļus.

### Augšējās atmiņas bloki (Upper Memory Bloks)

UMB apgabalā ir brīvi atmiņas bloki, kurus var izmantot procesori sākot ar 80386. Kopumā izmantotā tehnoloģija sakrīt ar EMS būtību, bet ir divas atšķirības:

1) šajā atmiņā var ielādēt kā datus, tā arī programmas. Programmām nevajag rūpēties par pieeju UMA brīviem blokiem, jo šo f-ju veiksmīgi izpilda DOS.
2) Nevar izmantot atmiņu, kas ir lielāka par UMB reģioniem, jo lpp aizvietošanas mehānisms šeit nestrādā. UMB vadību izpilda kopīgi abi draiveri. Par cik UMA atmiņai raksturīga programmēšana, tad UMB reģionus izmanto draiveru ielādēšanai.

## Atmiņas vadības līdzekļu izmantošana

Lai nodrošinātu atmiņas pilnas struktūras atbalstīšanu ir nepieciešams:

a) pieslēgt draiveri HIMEM.SYS
b) pieslēgt draiveri EMM386.EXE ar atslēgu RAM
c) failā CONFIG.SYS uzrādīt komandu DOS=HIGH, UMB
d) ar komandu DEVICEHIGH var uzrādīt ārējo draiveru ielādēšanu augšējā atmiņā, bet ar komandu LOADHIGH rezidentkomandu ielādēšanu augšējā atmiņā.

**RAMDRIVE.SYS** – šo draiveri izmanto virtuālo disku organizēšanai operatīvajā atmiņā. Šo paņēmienu izmanto, lai paātrinātu datu apmaiņu starp RAM un diskatmiņu. Ar atslēgām komanda var uzrādīt RAM apjomu, ko izmanto diskam, sektora izmēru, failu skaitu diska saknes katalogā un virtuālā diska organizēšanas veidu(XMS specifikācija vai EMS specifikācija).

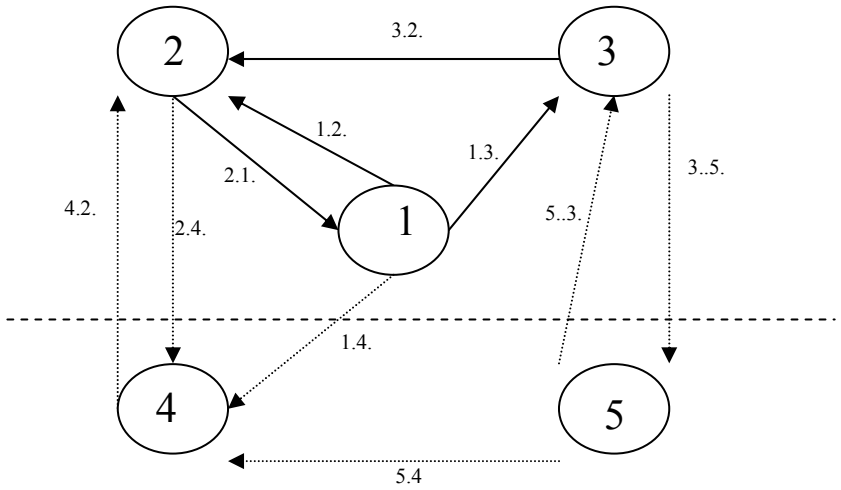
**SMARTDRV.EXE-** šo draiveri izmanto kešatmiņas organizēšanai paplašinātā atmiņā, lai paātrinātu I/O operācijas. Par diska kešēšanu sauc visbiežāk izmantojamo diska sektoru glabāšanu RAM. Izmantojot kešu notiek informācijas apmaiņa starp disku un RAM.

# 6. Lekcija

## Laiksakritīgie procesi.

Programma ir komandu statiska secība. Process ir darbību secība, tas ir, dinamisks objekts. Process ir programma izpildīšanas stadijā.

Eksistēšanas gaitā process var mainīt savu stāvokli.




1.-izplatīšanās;
2.-gatavība;
3.-sabloķēšanās;
---
4.-apturēts – gatavs;
5.-apturēts – sabloķēts;

**Procesa stāvokļu diagramma**

Process atrodas izpildīšanas stāvoklī, ja procesors tiek iedalīts procesam un atrodas procesa rīcībā. Process atrodas gatavības stāvoklī, ja varētu izpildīties uzreiz pēc procesora nodošanas viņa rīcībā. Process atrodas sabloķēšanas stāvoklī, ja viņš gaida, kad notiks kaut kāds notikums, lai būtu iespējams turpināt savu darbu.

1.2. – process tiek pārcelts gatavības stāvoklī, ja ir beidzies procesam iedalītais laika kvants līdz tam momentam, kad process brīvprātīgi atbrīvo procesoru.

1.3. – process tiek pārcelts sabloķēšanas stāvoklī, ja viņš inicializē ievad un izvad procedūras un gaida to pabeigšanu.

3.2. – process tiek pārcelts gatavības stāvoklī, kad ir noticis notikums, kuru process gaidīja.

2.1. – process sāk izpildīties, kad dispečers izdala viņam procesoru.

Lai varētu efektīvi regulēt datora resursu slodzi, OS var apturēt un atjaunot procesus. Apturēšanu izmanto, lai uz neilgu laiku izslēgtu procesu, kad sistēma ir pārslogota. Ja apturēšana ir paredzēta uz ilgāku laiku, procesam jāatbrīvo sistēmas resursi. Atjaunošana ir operācija, kura sagatavo procesu atkārtotai palaišanai no tā paša punkta, kurā process bija apturēts.

3.5. – ir vajadzīga, lai varētu ātrāk aktivizēt sabloķēto procesu, jo apturēšanas operācijai ir augstāka prioritāte.

Procesu vadībai OS izmanto procesa vadības bloku (**PCB** - Process Control Block), kuros tiek saglabāta visa informācija par procesu, piemēram, tekošais stāvoklis, prioritāte, izdalītie resursi, utt.

OS tiek realizēti mehānismi, kuri ļauj izpildīt dažādas procesa operācijas, tādas kā procesa izvietošanas, iznīcināšana, prioritātes noteikšana un mainīšana, palaišana, utt.

# Asimptomie procesi

Laiksakritīgie procesi ir tādi, kuri eksistē un izpildās vienlaicīgi. Tādi procesi var darboties pilnīgi neatkarīgi viens no otra. Sinhronizācija ir darbība, kas nodrošina noteiktu notikumu sakrišanu laikā 2 vai vairāku asinhronu procedūru izpildes gaitā.

Sinhronizācija ir procesa uzsākšanas momenta piekārtošana kādam noteiktam notikumam sistēmā.

No citas puses procesi vienmēr konkurē, lai izmantotu ierobežotus resursus, tādus kā centrālais procesors, atmiņa, faili. Sadarbības un konkurences elementi prasa, lai būtu sakarība starp procesiem. Sfēras, kur sakarības starp procesiem ir ļoti svarīgi var klasificēt sekojoši:

## savstarpējā izslēgšana

savstarpējās izslēgšanas problēma ir tāda, lai nodrošinātu pieeju koplietojamiem resursiem, tikai vienam procesam dotajā momentā. Sistēmas resursus var klasificēt kā koplietojamos, ja tās vienlaicīgi var izmantot vairāki procesi. Pārējos resursus sauc par resursiem, ko nevar izmantot vienlaicīgi. Ja process griežas pie koplietojamiem resursiem, tad uzskata, ka process atrodas savā kritiskajā apgabalā. Ja procesi sadarbojas izmantojot kopīgus manīgos un viens no procesiem atrodas savā kritiskajā apgabalā, tad pārējiem procesiem tiek izslēgta iespēja iziet savos kritiskajos apgabalos.

## sinhronizācija

sinhronizācija ir viena no izpildīšanas ātruma operācijām, nav pareģojams salīdzinoši ar cita procesora ātrumu, jo tas ir atkarīgs no pārtraukuma frekvences un no tā cik bieži un uz cik ilgu laiku katrs no procesiem aizņem procesoru. Var teikt, ka procesi izpildās asinhroni viens pret otru. Lai sasniegu veiksmīgu procesu sadarbību var uzrādīt svarīgus punktus, kuros procesiem jāsinhronizē savas darbības. Ir tādi punkti, kuros process nevar uzsākt savu darbību, kamēr cits process beigs savu darbību. OS ir atbildīgas par to, lai nodrošinātu tāda veida sinhronizēšanas mehānismu.

## strupceļš

strupceļš ir situācija, kad procesi, kas pieprasa vienus un tos pašus resursus bloķē viens otru. Lai novērst šādas situācijas vajag samazināt to negatīvo ietekmi un tā arī ir viena no OS funkcijām.

## Semafori

Ļoti svarīgu ieguldījumu starpprocesu komunikācijas izdarīja Deikstra, kas piedāvāja semafora koncepciju un primitīvas operācijas *wait* un *signal*, kas ietekmē uz semaforiem.

Semaforas ir īpašs datu tips, ko izmanto, lai sinhronizētu vairākus procesus, kas izpildās vienlaicīgi. Semafors ir aizsargāts mainīgs, kuram vērtību var nolasīt un nomainīt izmantojot speciālos operācijas *wait* un *signal* un semafora inicializēšanas komandas.

wait(S) : IF val(5)>0 THEN val(5):=val(5)-1 ELSE (gaidīt uz S)

signal(S) : IF (process gaida uz S) THEN (atļauts procesa darbu turpināt) ELSE val(5):=val(5)+1

Var redzēt, ka katra operācija *signal* palielina semafora vērtību uz 1, bet katra veiksmīga operācija *wait* pazemina semafora vērtību. Tātad semafora vērtība tiek noteikta ar sekojošo izteiksmi:

val(S) = c(S) + ns(S) – nw(S), kur

S – semafors;
val(S) – semafora vērtība;
c(S) – semafora inicializēta vērtība;
ns(S) – operāciju *signal* skaits;
nw(S) – operāciju *wait* skaits.

val(5) >= 0 $\rightarrow$ nw(S)<=ns(S) + c(S)

Visbiežāk rinda uz semaforu tiek apkalpota atbilstoši FIFO disciplīnai, kaut gan var organizē arī citu algoritmu. Semaforus un operācijas *wait* un *signal* var realizēt kā ar programmas tā arī ar aparatūras līdzekļiem. Parasti tie tiek realizēti sistēmas nodaļā, kur notiek procesu stāvokļu maiņas vadība.

Izmantojot semaforu koncepciju var atrisināt izslēgšanas problēmu ar vienkāršu paņēmienu, tas ir, katru procesa kritisko apgabalu ielenkt ar operācijām *wait* un *signal* uz vienīgā semafora ar sākuma stāvokli "1".

Semaforu var izmantot sinhronizēšanas mehānisma realizēšanai – viens process bloķē pats sevi izpildot operāciju *wait* uz semafora ar sākuma stāvokli "0", lai sagaidītu kāda notikuma pienākšanu. Cits process konstatē, ka gaidāmais notikums ir noticis un atjauno sabloķēto procesu ar operācijas *signal* palīdzību. Pieņemsim, ka process A nevar turpināties tālāk par punktu L1, kamēr process B nesasniegs punktu L2. Šajā piemērā sinhronizācija ir asimetriska, jo procesa A turpināšana tiek regulēta ar procesu B. Piemērs, kad katrs procesiem regulē cita procesa izpildīšanos ir klasiska problēma *ražotājs – patērētājs*. Ražotājs sadarbojas ar patērētāju izmantojot buferi, kurā ražotājs ievieto elementus un no kura patērētājs izņem šos elementus. Buferī ir ierobežota ietilpība. Šajā piemērā ir 2 sinhronizācijas problēmas:

- ražotājs nevar ievietot elementus buferī, ja buferis ir aizpildīts;
- patērētājs nevar izņemt elementus no bufera, ja buferis ir tukšs.

Vēl viena savstarpējās izslēgšanas problēma, tas ir, buferi vajag aizsargāt no dažādu procesu vienlaicīgu pieeju tām.

# 7. Lekcija

## Laiksakritīgie procesi

## Monitori

Monitors ir programma, kurā sistēma kontrolē, vada un pārbauda operāciju izpildi. Monitors ir procedūru un informatīvo struktūru kopa, kuru procesi izmanto laikdalības režīmā un kuru katrā momentā var izmantot tikai viens process.

Salīdzinot ar semaforiem, monitoriem ir sekojošas priekšrocības:

- Monitors ir ļoti elastīgs līdzeklis, ar kura palīdzību var realizēt gan semaforus, gan citus sinhronizācijas līdzekļus.
- Visu sadalāmo mainīgo lokalizēšanu monitorā ļauj atbrīvoties no sarežģītām konstrukcijām sinhronizējošos procesos.
- Monitori ļauj procesiem kopīgi izmantot programmu, kura pēc būtības ir kritisks apgabals.

Monitora piemērs ir programma plānotājs, kas sadala kaut kādus resursu. Katru reizi, kad process grib saņemt savā rīcībā resursa daļu tas griežas pie plānotāja. Šo plānotāju izmanto visi procesi un katrā no procesiem, jebkurā momentā, var rasties nepieciešamība griezties pie plānotāja, bet plānotājs nevar apkalpot vienlaicīgi vairāk kā vienu procesu. Ir situācijas, kad monitoram vajag apturēt procesu, kurš griežas pie tā, piemēram, ja process pieprasa resursu, kuru jau izmanto cits process. Šajā gadījumā monitors var sabloķēt šo procesu, kamēr resurss nebūs atbrīvots. Procesu sabloķēšanai izmanto operācijas – wait un signal, kad sabloķē procesu. Jāuzrāda arī nosacījums, kad process varēs turpināt darbu, piemēram, resurss brīvs, kad nosacījums tiek izpildīts monitors izstrādā signālu – signal un pasludina, ka nosacījums ir izpildīts, ja kaut kādi procesi gaida šā nosacījuma izpildi, tad viens no tiem pamostas un saņem atļauju turpināties.

```
monitor resursu_plānotājs
var resurss_aizņemts:boolean;
    resurss_brīvs:boolean;

procedure paķert_resursu
 begin
  if resurss_aizņemts then
  wait(resurss_brīvs);
  resurss_aizņemts:=true;
 end;

procedure atdot_resursu
 begin
  resurss_aiņemts:=false;
  signal(resurs_brīvs);
 end;
 begin
  resurs_aizņemts:=false;
 end;
```

tajās procedūrās var būt kļūdas, jo tas ātrumā ir spiests no slaida

Monitors ir viena vai vairākas procedūras ar statiskām, globālām, informatīvām struktūrām. Kad pie monitora griežas pirmo reizi visi mainīgie saņem savas sākuma vērtības, kad vēlāk tiek izmantotas mainīgo vērtības, kas palika no iepriekšējās griešanās. Mainīgie ar tipu condition nav līdzīgi parastajiem mainīgajiem. Kad tiek definēts tāds mainīgais, tas nozīmē, ka tiek organizēta rinda. Process, kas izpilda wait komandu tiek ievietots šajā rindā. Wait process, kas izpildīja operāciju signal ļauj citam procesam, kas gaidīja iziet no rindas un ieiet monitorā.

Ir formulēti strupceļu izcelšanās četri nepieciešamie nosacījumi:

1. Savstarpējās izslēgšanas nosacījums – procesi prasa piešķirt tiesības uz resursu monopola vadību.
2. Resursu gaidīšanas nosacījums – procesi patur jau izdalītus to rīcībā esošus resursus un gaida papild resursu izdalīšanu.
3. Resursu sadalīšanas no jauna neiespējamības nosacījums – nevar atņemt resursus no procesiem, kamēr tie nebūs izmantoti, lai process varētu pabeigt darbu.
4. Riņķa gaidīšanas nosacījums – eksistē procesu riņķa virkne, kurā viens vai vairāki procesi patur vienu vai vairākus resursus, kuri ir nepieciešami nākamajam virknes procesam.

Attiecīgi pret strupceļiem, katrai sistēmai ir sava politika. Eksistē trīs veidu politikas:

1. Novērst strupceļus – ideja balstās uz tā, ka sistēma nedrīkst nonākt strupceļa stāvoklī. Tāpēc kad kaut kāds process izstrādā pieprasījumu, kas var novest sistēmu pie strupceļa, sistēma veic pasākumus, lai izvairītos no bīstamā stāvokļa – vai nu neapmierina pieprasījumu, vai atņem resursu no cita procesa. Metodes priekšrocība – strupceļš vienmēr tiek novērsts. Trūkumi – neefektīvi tiek izmantoti sistēmas resursi un dārgi var izmaksāt algoritms, kas realizē šo pieeju.
2. Automātiski atrast – pieļauj, lai sistēma nokļūtu strupceļa situācijā, kad tas reāli notiek sistēma ar speciālas programmas palīdzību atrod strupceļu un pēc tam var atņemt resursus no dažādiem procesiem, lai citi varētu izkustēties no vietas. Šī metode ļauj izmantot resursus efektīvāk un ja reāli strupceļa situācijas parādās diezgan reti, tad resursu noslogojuma palielināšana sedz izdevumus strupceļa atklāšanai un izejai no tās.
3. Likvidē ar operatora palīdzību – paredz, ka strupceļa situācija rodas ļoti reti, lai par to uztrauktos. Ja tomēr strupceļš rodas tad sistēmu vienkārši palaiž no jauna. Bet var gadīties, ka sistēmas palaišana no jauna var izmaksāt ļoti dārgi.

OS visbiežāk izmanto pirmo metodi. Vispopulārākais algoritms ir baņķiera algoritms, ko piedāvāja Deikstra. Šo algoritmu izskatīsim, ka **n** procesi izmanto tikai viena veida resursus un sistēma var garantēt šī resursa fiksēta skaita **t** sadalīšanos pa procesiem. Jau iepriekš ir zināms, resursu maksimālais skaits, kas ir vajadzīgs katram procesam. Sistēma var ierosināt procesu, ja $\mathbf{m(i) \le t}$. Process var gaidīt papildus resursu izdalīšanu un sistēma garantē, ka galu galā process to saņems. Resursu skaits ir izdalīts procesam, dotajā momentā $\mathbf{l(i) \le m(i)}$.

**c(i)=m(i)-l(i)** – procesa pašreizējā vajadzība pēc resursa.

$\mathbf{a = t - \sum_{1}^{n} l(i)}$ – resursu skaits, kas ir pieejams sadalīšanai dotajā momentā.

Ja process saņem savā rīcībā visus tam vajadzīgos resursus tas garantē, ka pēc kāda laika pabeigs savu darbu un atgriezīs resursus sistēmai. Sistēmas stāvoklis ir drošs, ja OS var nodrošināt visu procesu pabeigšanu pārskatāmā laika periodā.

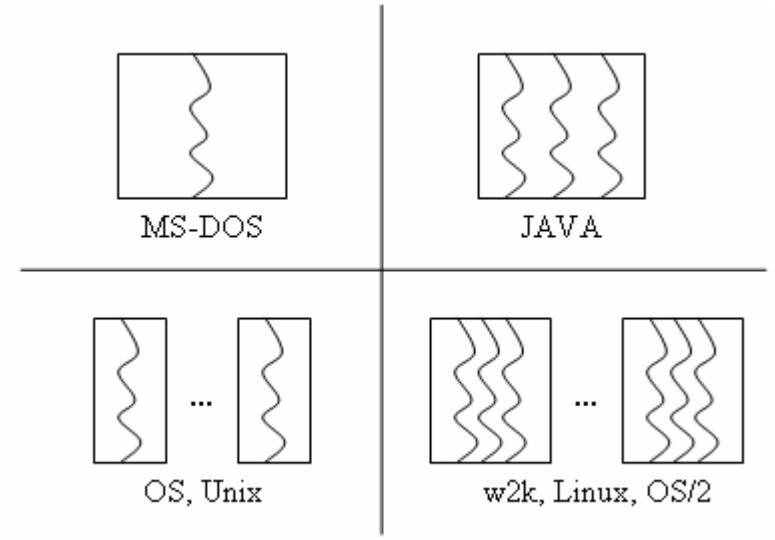
Man jāatvainojas, bet te jābūt zīmējumam no 7. lekcijas;)

## Plūsmas

(Kaut kādu) koncepciju var raksturot ar diviem parametriem:

1. resursu pārvaldīšana (resource ownership)
2. plānošana un izpildīšana (sheduling/execution)

OS var uzskatīt šos raksturojumus vienu no otra atsevišķi. Un šajā gadījumā dispečerizācijas vienību (parametru) sauc par plūsmu. Daudz plūsmu režīms ir OS spēja uzturēt vairāku plūsmu izpildīšanu viena procesa ietvaros.




Visas procesa plūsmas sadalās savā starpā. Procesa resursi atrodas vienā adrešu platībā un var izmantot pieeju vieniem un tiem pašiem datiem, ja viena plūsma maina kādus datus atmiņā tad citās plūsmās jau var novērot izmaiņas. Ja viena plūsma atver failu lasīšanai citas plūsmas arī var lasīt šo failu.

No ražīguma viedokļa plūsmu izmantošanai ir šādas priekšrocības:

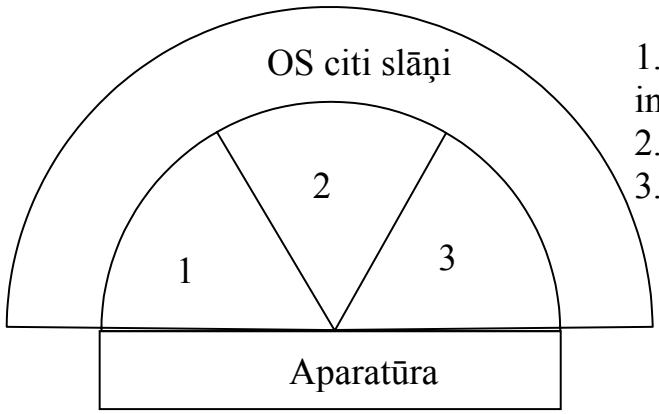
- jaunas plūsmas izveidošana esoša procesa ietvaros prasa mazāk laika nekā jauna procesa izveidošana;
- plūsmu var pabeigt ātrāk nekā procesu;
- pārslēgšanās starp plūsmām notiek ātrāk nekā pārslēgšanās starp procesiem;
- tiek paaugstināta informācijas apmaiņas efektivitāte.

# 8. Lekcija

## OS kodols

## OS kodola sastāvdaļas

Kodols ir OS programmu kopums, kas veido tās centrālo daļu un realizē datora vadības funkcijas. Kodols ir OS neliela daļa, kura tiek izmantota ļoti intensīvi un kuras kods pastāvīgi atrodas pamatatmiņā. Kodols nodrošina interfeisu starp datora aparatūru un OS pārējām sastāvdaļām.




1.First – level interupt handler
2.Dispatcher
3.WAIT, SIGNAL procedūras

Galvenais kodola mērķis ir nodrošināt visu saistīto ar procesiem un operāciju izpildīšanu, tas ir, pārtraukumu apstrāde, procesu pārslēgšana starp procesiem un starpprocesu komunikāciju mehānismu implementēšanu. Var uzskatīt, ka kodols sastāv no trīs daļām:

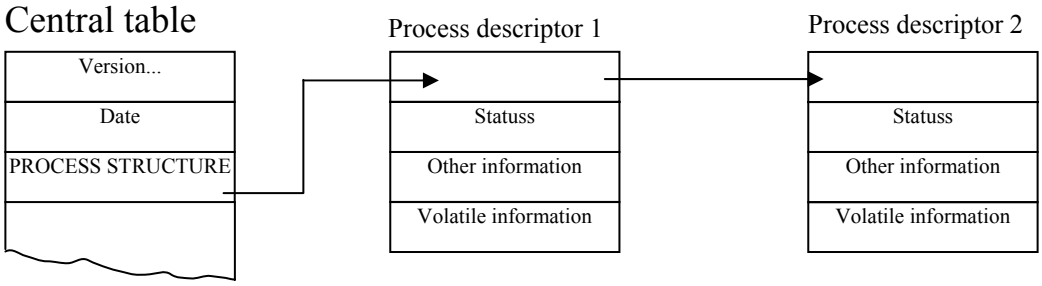
- pirmā līmeņa pārtraukumu apdarinātājs
- dispečers
- divas procedūras (rutīnas), kas implementē signālus WAIT un SIGNAL

OS citi slāņi – tie ir atmiņas vadība, I/O organizācija, failu sistēmas plānošana un resursu iedalīšanas un aizsardzības mehānismi.

Procesu attēlošana

Visas kodola sastāvdaļas funkcionē izmantojot dažādas datu struktūras. Katru procesu var raksturot ar procesa deskriptoru, kas satur visu nepieciešamo informāciju par procesu. No sākuma var pieņemt, ka procesa deskriptors satur procesa identifikatoru, informāciju par procesa statusu (jeb stāvokli),energoatkarīgu informāciju (dažādu reģistru vērtības) u.c. informāciju.

Visu procesu deskriptori ir iekļauti procesu struktūrā – tas ir, saistītā sarakstā. Tā ir pirmā datu struktūra, ar kuru strādā OS un kura ir iekļauta centrālā tabulā.

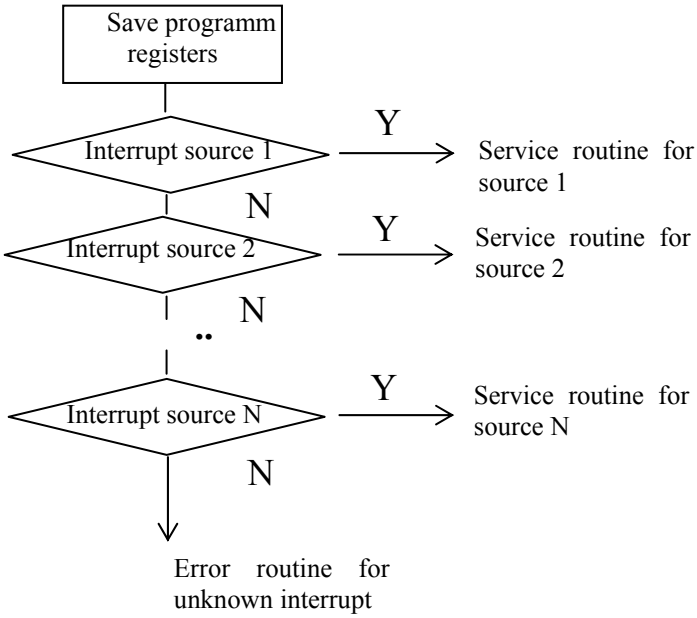



Centrālāas tabulas mērķis ir nodrošināt pieeju visām OS datu struktūrām. Tabula satur rādītājus uz katru struktūru un dažādu globālu informāciju, tādu kā, OS versiju, datumu.

## Pirmā līmeņa pārtraukumu apstrādātājs (FLIH)

FLIH ir OS daļa, kas atbildīga par pārtraukumu apstrādi. Tas ir:

- Noteikt pārtraukumu avotu
- Inicializēt pārtraukumu apstrādi




Pārtraukumu avota noteikšanai izmanto iekārtu stāvokļa karodziņu testēšanai.

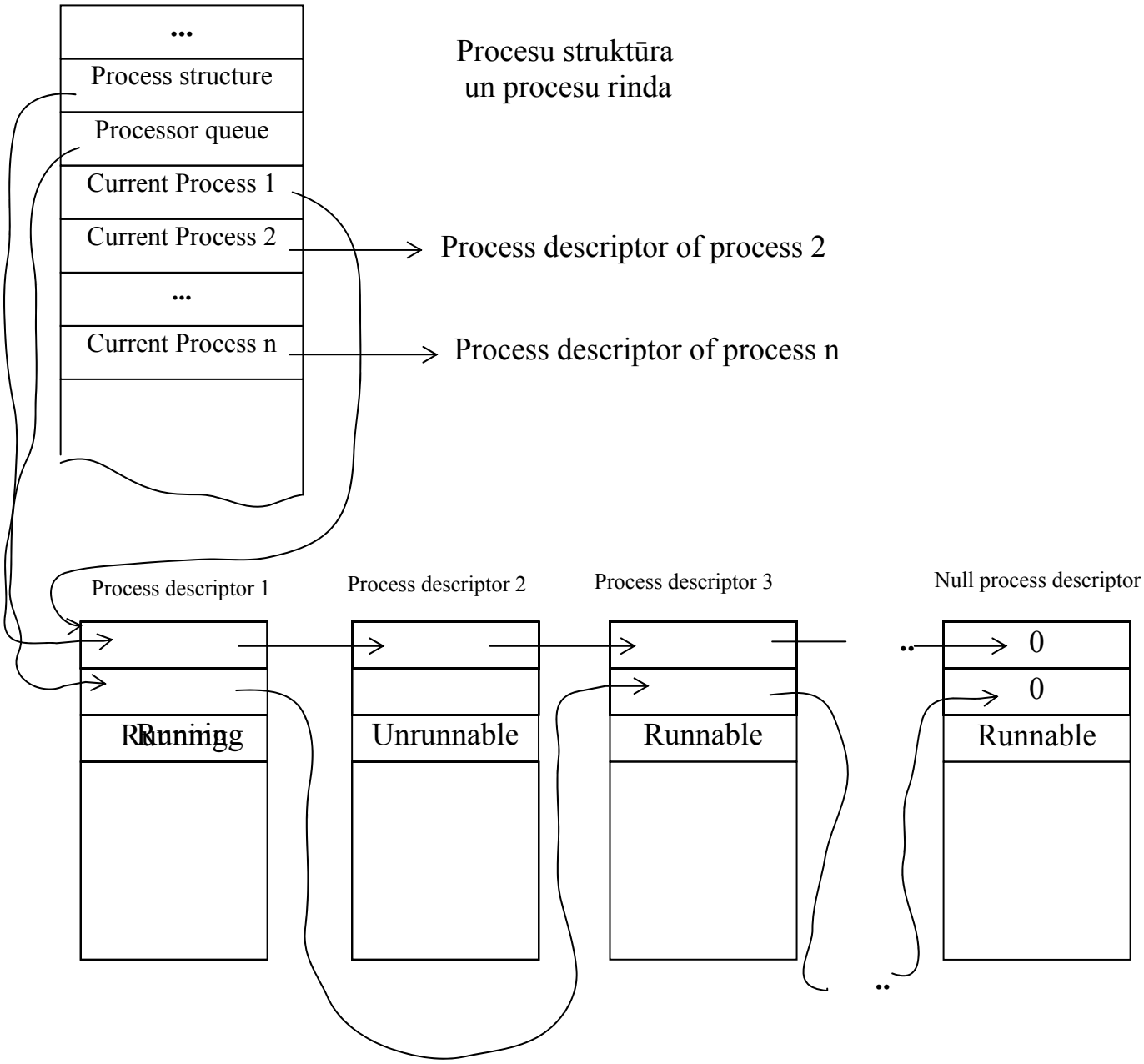
Karodziņš – mainīgais, kura vērtība norāda, ka ir sasniegts kāds noteikts iekārtas stāvoklis vai izpildās programmas nosacījums.

Pārtraukumu avota pārbaude tiek organizēta pārlēcieniem virknes veidā. Pie tam visbiežāk sastopamie pārtraukumu avoti atrodas virknes sākumā.

Otra FLIH funkcija nozīmē, ka notiek pārtraukumu apkalpošanas rutīnas izsaukšana un izpilde. Svarīgi atzīmēt, ka pārtraukuma izcelšana vienmēr ietekmē dažādu procesu stāvokļus, piemēram, process, kurš bija sabloķēts sakarā ar to, ka gaidīja I/O operācijas pabeigšanu, lai saņemtu to savā rīcībā, var kļūt par izpildāmo pēc pārtraukuma apstrādes. Visos gadījumos, kad notiek procesa stāvokļa mainīšana, tas ir saistīts ar rutīnas iedarbību uz lauku status to procesu deskriptoros, kas ir saistīti ar šo pārtraukumu. Ar procesora pārslēgšanu starp procesiem nodarbojas cita koda daļa – dispečers.

## Dispečers

Tā ir OS programma, kas izvieto datus vai uzdevumus izpildīšanai. Dažreiz dispečeru sauc arī par zema līmeņa plānotāju.




Dispečers vienmēr ir iedarbināts pēc pārtraukumiem. Dispečera darbības ir sekojosas:

1) Pārbaudīt, vai dotajā procesā tekošais process ir vispiemērotākais izpildīšanai. Ja tas ir tā, tad atgriež šim procesam kontroli, ja nē, tad
2) Saglabā tekošā procesa energoatkarīgu informāciju procesa deskriptorā
3) Atrod vispiemērotāko izpildīšanai procesu un šī procesa energo atkarīgu informāciju pārceļ no procesa deskriptora uz reģistriem.
4) Pārslēdz procesoru uz izvēlēto procesu atbilstoši saglabātajam punktam.

Vispiemērotāko izpildīšnai procesu var noteikt ņemot vērā procesa prioritāti, ko nosaka augsta līmeņa plānotājs, ņemot vērā resursu vajadzības, laika intervāu, kuru

jau izmantoja process un uzdevuma svarīgumu. Visus gatavus izpildīšanai procesus deskriptors sakārto rindā atbilstoši procesa prioritātei, šajā rindā vispiemērotākais izpildīšanai process atrodas rindas sākumā.

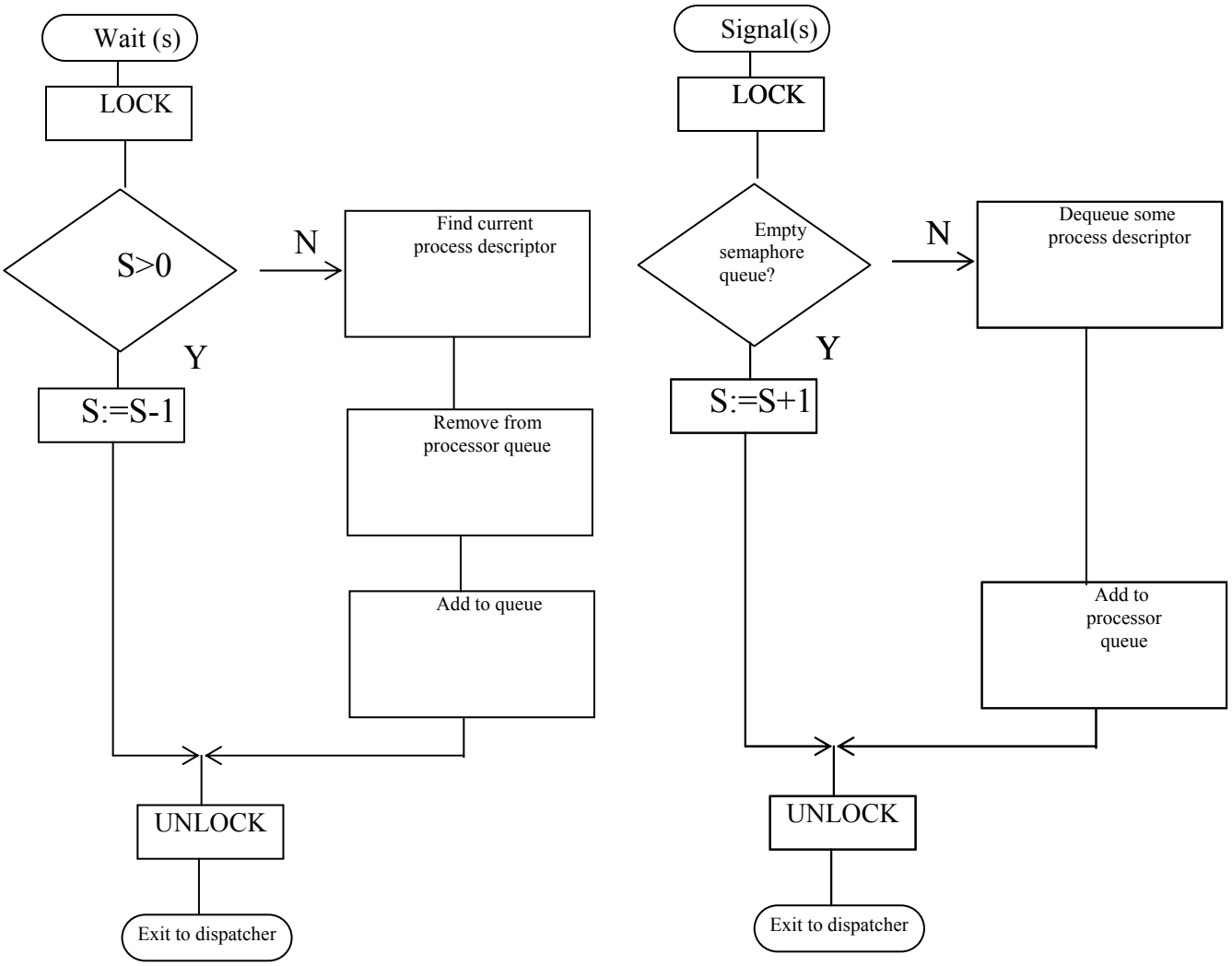
Tātad dispečera loma ir aktivizēt pirmo rindas procesu, kas dotajā momentā neizpildās un cita procesora.

Ja augsta līmeņa plānotājs strādā neefektīvi, varētu gadīties situācija, kad procesoram nav darba. Tāpēc rindas beigās ir paredzēts ekstra process (nullprocess), kuram ir viszemākā prioritāte, un kuru var vienmēr palaist uz izpildīšanu (piemēram, testēšanas programma).

## WAIT un SIGNAL implementēšana.

WAIT un SIGNAL ir starpprocesu komunikācijas mehānismi un tie ir iekļauti OS kodolā, jo:

- Tiem ir jābūt pieejamiem visiem procesiem.
- Operācija WAIT iedarbojas uz sabloķētiem procesiem un tāpēc tai jābūt pieejai dispečeram.
- Izpildot operāciju signal uz tā semafora, kur process izpildīja operāciju WAIT, var ļoti ērti pārcelt procesu stāvoklī izpildošs.



Tāpēc operācijai SIGNAL jābūt pieejai pārtraukumu rutīnām.

Visvieglāk procesu bloķēšanai un atbloķēšanai izmantot rindu, tas ir, kad process griežas pie semafora, kura vērtība ir nulle, viņš tiek iekļauts semafora rindā un būs bloķēts. Un otrādi, kad tiek izpildīta operācija SIGNAL, kāds no procesiem tiek paņemts no rindas un kļūst par izpildāmo. Visbiežāk semaforos izmanto rindas atbilstoši FIFO disciplīnai. Tas nozīmē, ka semaforu struktūrā jābūt veselā tipa mainīgajam, rindas rādītājam un vēl vienam elementam, kas atspoguļo rindas organizēšanu.

Operācijas WAIT un SIGNAL ir tādas, kuras vienā un tajā pašā laikā momentā var izpildīt tikai viens process. Tas nozīmē, ka procedūras realizē WAIT un SIGNAL operāciju ar kādu atslēgšanas operāciju un jābeidzas ar kādu atslēgšanas operāciju.

Var izmantot speciālus mainīgos (karodziņus), kas nosaka, vai procesam ir atļauta ieeja procedūrā WAIT vai SIGNAL. Piemēram, ja karodziņu vērtība nav negatīva, ieeja atļauta, citādi nē. Tādu pieeju realizē IBM datoros.

Šādas procedūas (WAIT un SIGNAL) sauc par ekstra kodiem, un tie ir daļa no visu procesu instrukcijām.

## Atmiņas pārvaldība.

Atmiņas pārvaldība – tas ir metožu kopums, kas nodrošina efektīgu atmiņas resursu vadīšanu un iedalīšanu. Atmiņas vadības mērķi ir šādi:

1. Nodrošināt pārvietojamību (Relocation)

Pārvietojamība – tā ir iespēja ielādēt programmu jebkurā pieļaujamā OS atmiņas vietā, kā arī bez modificēšanas pārvietot to no vienas atmiņas apgabala uz otru.

2. Nodrošiāt atmiņas aizsardzību (Protection)

Atmiņas aizsardzība – tas ir aparatūras ar programmatūras līdzekļiem realizēts mehānisms, kas nodrošina kontrolējamu pieeju visai atmiņai un tās atsevišķām daļām, ko izmanto dažādi procesi.

3. Nodrošināt atmiņas dažu apgabalu koplietošanu (Sharing)

Jānodrošina kontrolējama pieeja atmiņas apgabalam, ko vienlaicīgi izmanto dažādi procesi.

4. Nodrošināt atmiņas loģisku organizēšanu

pārsvarā visas programmas ir sadalītas segmentos. No šādas pieejas var iegūt šādas priekšrocības:

- var kodēt visus segmentus neatkarīgi un visu attiecību starp sgmentiem realizēt izpildīšanas gaitā.
- Ir iespējams piešķirt segmentiem aizsardzību dažādas pakāpes (piemēram, tikai lasāms, tikai izpildāms)
- Var izstrādāt mehānismu segmenta koplietošanai.

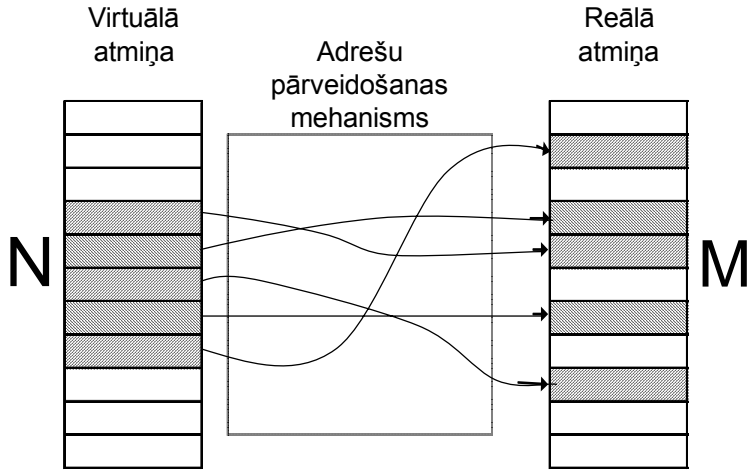
5. Nodrošināt atmiņas fiziskās izmantošanas organizēšanu (Physical organization)

OS jānodrošina informācijas pārvietošana starp atmiņas līmeņiem.

# 9. Lekcija

## Virtuālā atmiņa.

Visus iepriekš minētos mērķus var sasniegt izmantojot konceptuāli vienkāršu mehānismu, t.i., adreses translāciju (address map). Adreses translācija ir instrukcijas vai datu elementa adreses pārveidošanas process, kura rezultātā tiek iegūt adrese elementa ielādēšanai operatīvajā atmiņā. Tā ir OS funkcija, kuru var raksturot sekojoši f: N→M, un kur N ir programmas adrese, bet M atmiņas adrese, kur programma tiek izvietota.




Virtuālā adrese ir virtuālas atmiņas sistēmas adrese, kas programmas izpildes gaitā tiek pārveidota reālās atmiņas adresē.

Virtuālā atmiņa tas ir atmiņas apgabals, ko datora lietotājs var uzskatīt kā adresējamu operatīvo atmiņu, ar kurām virtuālās adreses pārveido reālās adresēs. Virtuālās atmiņas apjomu nosaka datora adresēšanas shēma un palīgatmiņas apjoms.

### Virtuālās atmiņas implementēšana.

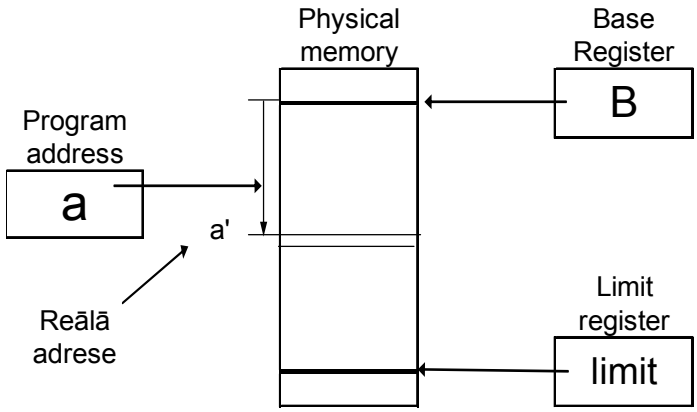
Virtuālās atmiņas realizēšanas un vadības metodes tika izstrādātas pagājuša gadsimta 60. gados. Izpētes bija tik veiksmīgas, ka praktiski netika izmainītas kopš tā laika.

Virtuālas metodes implementācijas veidi:

1. Bāzes un robežas reģistrs (base and limit registers)
2. Lapošana (paging)
3. Segmentēšana (segmentation)
4. Iepriekšējo metožu kombinācija.

## Bāzes un robežas reģistrs

Lai sasniegtu pirmo un otro atmiņas vadības mērķus, var izmantot adrešu pārveidošanai bāzes un robežas reģistrus. Kad process tiek ielādēts atmiņa, bāzes reģistrā tiek saglabāta izvietošanas zemākā adrese, bet robežas reģistrā iedalītās atmiņas apjoms.




Adrešu pārveidošana notiek pēc sekojoša algoritma:

a) If a<0 or a> limit then atmiņas kļūda
b) f(a) →a':=B+a

Reģistrus neizmanto katram procesam, bet izmanto katram procesoram, uz kura process tiek izpildīts. Šo reģistru vērtība ir daļa no procesa destruktora energoatkarīgas informācijas. Šī adresēšanas shēma paredz, ka virtuālo adrešu diapazons ir mazāks vai vienāds reālo adrešu diapazonam. Ja programmētāju rīcībā vajag izdot virtuālo atmiņu, kas ir lielāka par reālo atmiņu, tad jāizmanto citas adresēšanas shēmas.

## Lapošana

Lapošana ir lappušu pārsūtīšana datorā starp reālo un virtuālo atmiņu. Lappuse ir fiksēta garuma programmas datu bloks, kam ir virtuālā adrese un kas var atrasties jebkurā iekšējās atminās apgabalā. Lappusi var pārsūtīt kā vienotu veselumu.

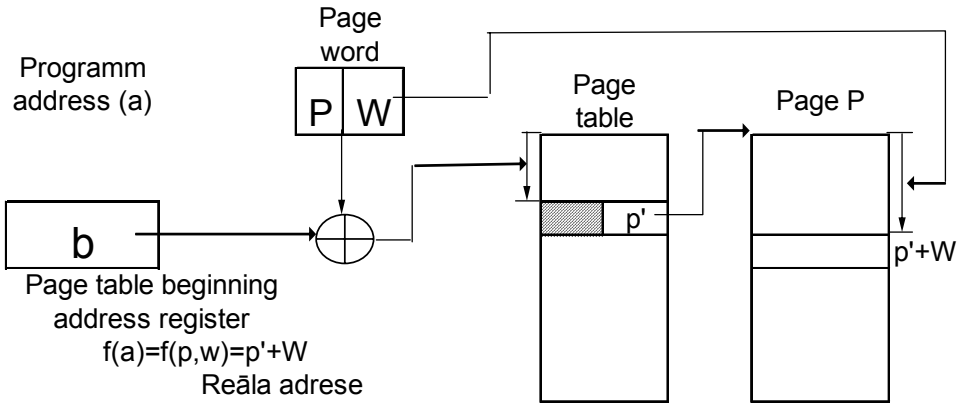
Lapošanas mehānisms nozīmē, ka virtuālo adrešu apgabals ir sadalīts viena izmēra lappusēs u n pamatatmiņa arī ir sadalīta tāda paša izmēra lappušu kadros ( page frames).

Procesam, kas izpildās atbilst aktīvās lappuses, kas dotajā momentā atrodas pamatatmiņā, un neaktīvās lappuses, kas atrodas sekundārā atmiņā. Lapošanas mehānisms realizē 2 funkcijas:

a) izpilda adreses veidošanas operācijas, nosaka kādai lappusei pieder programmas adrese un meklē atbilstošas lappuses.
b) Pārvieto lappuses no sekundārās atmiņas uz primāro un otrādi.

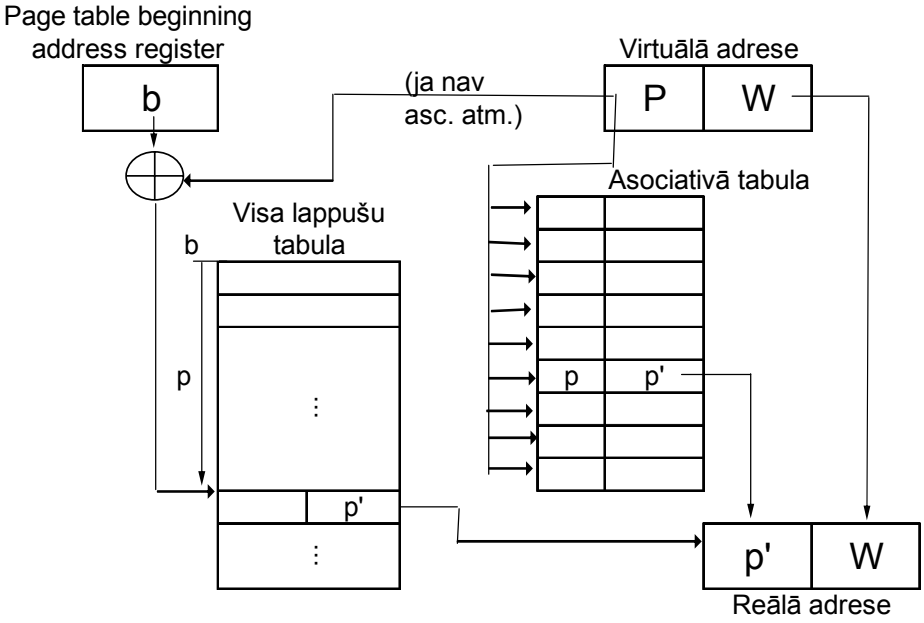
Lai realizētu pirmo funkciju uzskata, ka virtuālās adreses vecākie biti tiek interpretēti kā lappuses numurs un pārējie biti kā vārda numurs lappuses ietvaros. Adreses pārveidošanai izmanto lappušu tabulu ( Page table), kas raksturo kāda lappuses kadrā atrodas virtuālā adrese un vai atbilstošā lappuse atrodas pamatatmiņā.

Ja lappuse neatrodas pamatatmiņā, tad to vēl vajag pārvietot no sekundāras atmiņas, un ja brīvo lappušu kadru nav, tad vēl nepieciešams atrisināt uzdevumu, kuru lappusi pārvietot no primārās atmiņas uz sekundāro.




Praktiskā realizēšanā jāņem vērā, ka lappu1su tabula arī atrodas atmiņas konkrētajā apgabalā un tā sākuma adresi B saglabā speciālā reģistrā. Lai paātrinātu adrešu dinamisku pārveidošanu izmanto asociatīvo atmiņu:

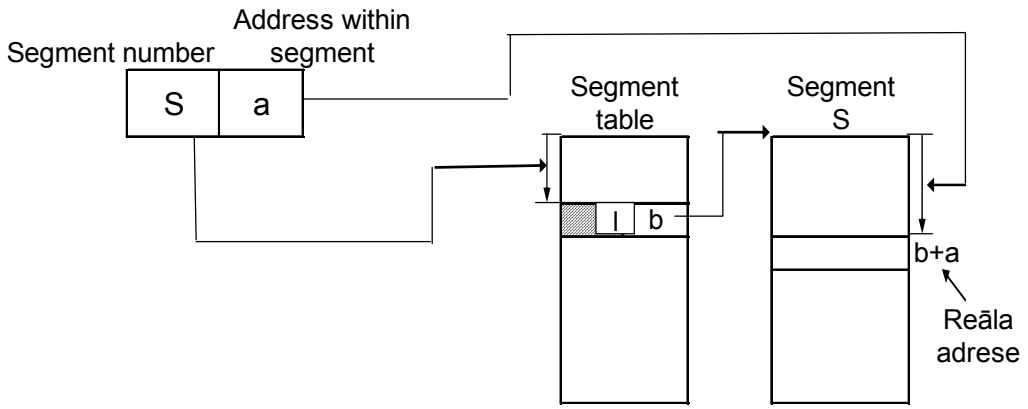
Asociatīva atmiņa ir tāda, kurā pieeju datiem realizē norādot datu lauka saturu.. Pārsvarā tādu atmiņu realizē ar reģistru izmantošanu. Asociatīvā atmiņa sastāv no lappušu adreses reģistriem, katrs no kuriem saglabā aktīvās lappuses numuru. Asociatīvā atmiņa ir maza un tāpēc tā nevar nodrošināt tik daudz reģistru cik lappušu ir vajadzīgs dotajā momentā. Tādēļ izmanto kombinēto shēmu, kad asociatīvajā atmiņā tiek izvietoti tikai aktīvāko lappušu numuri, bet pārējie tiek adresēti saskaņā ar lappušu tabulu.




Tādā gadījumā no sākuma tiek meklēta lappuse asociatīvajā atmiņa un, ja tās tur nav, tad kopējā lappušu tabulā, kur atrodas visas lappuses.

## Segmentācija

Adrešu platība tiek sadalīta segmentos, katrs no kuriem atbilst konkrētai procedūrai programmas modulī vai datu blokā. Segmentu izmēri ir dažādi, katru segmentu raksturo ar segmenta numuru S un garumu l.




Virtuālā adrese sastāv no 2 daļām, tas ir segmenta numurs S un adrese segmenta ietvaros a

Adreses pārveidošana segmenta ietvaros notiek sekojoši

a) Saņem programmas adresi V (V=(S, a)),
b) Izmanto S vērtību segmenta tabulas indeksēšanai,
c) Ja a ir mazāks nekā 0 vai lielāks nekā l, tad ir atmiņas kļūda,
d) Reālā adrese ir b+a.

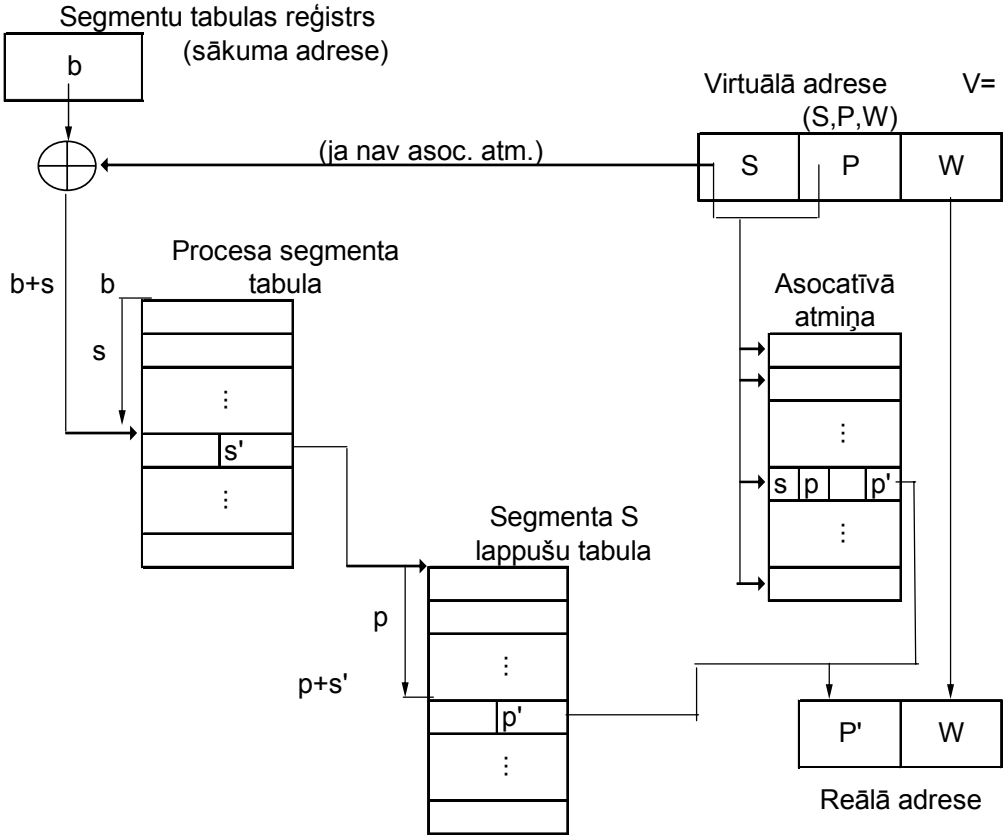
Atmiņas aizsardzība tiek realizēta ar adreses a salīdzināšanu ar segmenta garumu l. Palīg aizsardzībai var izmantot speciālus bitus, kas raksturo segmenta izmantošanas veidu (lasīt, rakstīt, papildināt utt.)

Lapošanas paņēmiens atšķiras no segmentācijas paņēmiena ar:

a) Lappuses izmērs ir fiksēts un to nosaka datora arhitektūra, bet segmenta izmēru nosaka lietotājs un tie var būt dažādi.
b) Programmas adreses salīdzināšana ar lappuses numuru un vārda numuru ir datora aparatūras funkcija un pārpildes funkcija vārda numura palielina lappuses numuru. Adreses salīdzināšana ar segmenta un vārda numuriem ir tīri loģiska un pārpildes situācija nepalielina segmenta numuru, bet izraisa atmiņas kļūdu.
c) Segmentācijas mērķis ir atmiņas platības loģiska sadalīšana, bet lapošanas mērķis ir atmiņas fiziska sadalīšana vien līmeņa atmiņas implementēšanai.

## Lapošanas un segmentēšanas kombinācija

Praksē lapošanas un segmentēšanas tehnoloģijas apvieno kas dod iespēju būtiski paātrina adrešu dinamisko pārveidošanu. Šajā gadījumā katrs satur lappuses veselo skaitu pie tam nav obligāti, lai segmentam visas lappuses atrodas primārajā atmiņā vienlaicīgi, un nav obligāti, lai virtuālās atmiņas blakus lappuses būtu blakus arī reālā atmiņā.



# 10. Lekcija

## Atmiņas iedalīšanas stratēģijas

Atmiņas iedalīšana ir darbība, kas saistīta ar atmiņas apgabalu piešķiršanu atsevišķiem procesiem. Var izdalīt 3 atmiņas iedalīšanas stratēģijas veidus:

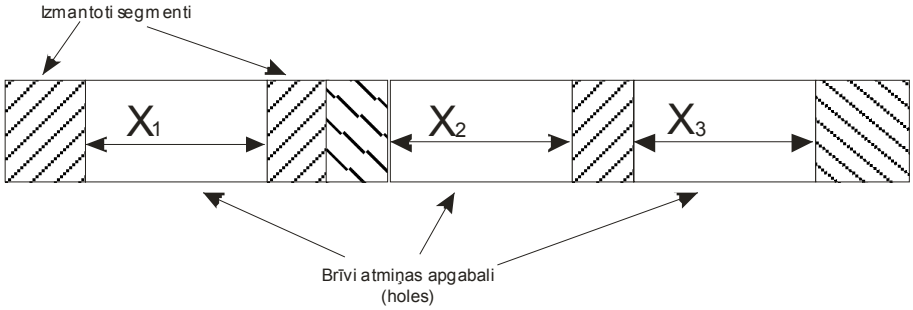
1) **Izstumšanas stratēģijas** (*replacement policies*). Mērķis – noteikt, kādu lappusi vai segmentu vajag pārcelt no pamatatmiņas uz sekundāro, lai atbrīvotu vietu citai informācijai.
2) **Iestumšanas stratēģijas** (*fetch policies*). Mērķis – noteikt, kad vajag pārrakstīt lappusi vai segmentu no sekundārās atmiņas uz primāro.
3) **Izvietošanas stratēģijas** (*placement policies*). Mērķis – noteikt vietu pamatatmiņā, kurā var izvietot lappusi vai segmentu.

Iestumšanas stratēģijas sistēmām ar lapošanas un segmentācijas mehānismiem praktiski neatšķiras, bet izstumšanas un izvietošanas stratēģijas atšķiras, jo lappušu izmērs ir vienāds visām lappusēm, bet segmentu izmēri ir dažādi.

### 1. Izvietošanas stratēģijas

Segmentēšanas mehānismam:

Sistēmās ar segmentāciju atmiņa var izskatīties sekojoši:




Brīvu apgabalu izmērs ir atšķirīgs, tāpat kā segmentu izmēri. Izvietošanas stratēģijas balstās uz brīvo apgabalu saraksta organizēšanas (*hole list)*. Šajā sarakstā tiek atspoguļota informācija par brīva apgabala izmēru $x_i$ un apgabala atrašanās vietu atmiņā. Stratēģijas uzdevums ir izlemt, kuru brīvo apgabalu izmantot un pēc tam reorganizēt sarakstu „*hole lis"*. Ja segmenta izmērs ir mazāks par brīvā apgabala izmēru $x_i$, tad segmentu var izvietot šajā brīvā apgabalā, bet, ja piemērotu apgabalu nav, tad vēl vajag atrisināt uzdevumu par brīvo segmentu apvienošanu.

Visbiežāk izmanto sekojošus algoritmus:

- **vispiemērotākais** *(Best fit)*. Brīvi apgabali sarakstā tiek sakārtoti pēc izmēriem augošā kārtībā. Saraksta sākumā atrodas mazākais brīvais apgabals : $x_1 \le x_2 \le ... \le x_n$. Tiek atrasts vismazākais apgabals $x_i$, kas apmierina šādu nosacījumu: $S \le x_i$, kur S – segmenta izmērs. Trūkums – atmiņā paliek daudz brīvu apgabalu ar ļoti mazu izmēru, kurus nevar izmantot segmentu izvietošanai un vajag bieži izpildīt defragmentēšanas operācijas.
- **Vismazāk piemērots** *(Worst fit)*. Brīvi apgabali sarakstā tiek sakārtoti pēc izmēra dilstošā kārtībā segmentu izvieto pirmajā apgabalā, kas atbilst nosacījumam $S \le x_i$. Priekšrocība, salīdzinot ar *Best fit* algoritmu – pēc

segmenta izvietošanas vismazāk piemērotākā apgabalā paliek brīvs apgabals, kuru var izmantot cita segmenta izvietošanai.

- **Pirmais piemērotais** *(First fit)*. Brīvo apgabalu saraksts tiek sakārtots šo apgabalu izvietošanas adrešu augošā kārtībā. Tiek izmantots pirmais no apgabaliem, kas atbilst nosacījumam $S \leq x_i$. Algoritma priekšrocība ir tāda, la ar laiku saraksta sākumā atrodas maiz pēc izmēra brīvi apgabali un vieglāk realizēt atmiņas defragmentēšanu.

Izvietošanas stratēģijas realizēšana sistēmās ar lapošanu ir daudz vieglāka. Fragmentēšana tādā veidā kā bija raksturots iepriekšminētajiem 3 algoritmiem vispār nav, jo visas lappuses un lappušu kadri pēc izmēra ir vienādi.

## 2. Izvietošanas stratēģijas ar lapošanu *(replacement policies).*

Stratēģijas uzdevums ir nolemt, kādus informācijas blokus pārvietot no primārās atmiņas uz sekundāro, kad ir nepieciešams izvietot jaunu informācijas bloku pamatatmiņā. Ideālā gadījumā ir vēlams pārvietot tādus blokus, kuri nebūs vajadzīgi vēl ilgu laiku. Diemžēl to nevar zināt precīzi un vienīgais, var izmantot zināšanas par bloku iepriekšējo uzvedību. Izstumšanas stratēģijas realizēšanai izmanto 3 algoritmus:

1) Izstumt visilgāk neizmantoto lappusi (LRU algoritms – *least recently used)*. Algoritma realizēšanai vajag glabāt informāciju par griešanās laiku katrai lappusei.
2) Izstumt lappusi, kuru izmantoja visretāk (LFU – *least frequently used)*. Šajā gadījumā vajag glabāt informāciju par griešanās intensitāti katrai lappusei. Trūkums ir tāds, ka lappusei, kas nesen bija izvietota pamatatmiņā ir zems intensitātes līmenis. Tas nozīmē, ka papildus vajag glabāt informāciju par laika intervālu, cik ilgi lappuse atrodas atmiņā.
3) Izstumt lappusi, kas pirmā bija ievietota pamatatmiņā. (FIFO – *First-In First-Out)*. Jāglabā informācija par to, kad lappuse ievietota pamatatmiņā. Trūkums ir tāds, ka varbūt šī lappuse ir visu laiku izmantošanā un pēc pārrakstīšanas sekundārā atmiņā atkal būs vajadzīga.

Sistēmās ar segmentāciju izstumšanas stratēģiju mērķis ir tāds pats kā sistēmās ar lapošanu, bet atšķirība ir segmentu izmēros. Ja vajag pārvietot pamatatmiņā nelielu segmentu, tad var izstumt arī nelielu segmentu no primārās atmiņas. Ja segments ir liels, tad jāmeklē pamatatmiņā lielu segmentu vai arī dažus mazus. Ja pēc segmenta izmēra ir dažādi varianti izstumšanai, tad var izmantot vienu no LRU, LFU vai FIFO algoritmiem.

## 3. Iestumšanas stratēģijas *(Fetch policies).*

Iestumšanas stratēģijas var sadalīt 2 klasēs – iestumšana pēc pieprasījuma (*Demand)*. Otra klase – iestumšana ar apsteidzi *(Antipatory)*.

- **Iestumšana pēc pieprasījuma** paredz, ka tiek ģenerēts pieprasījums un izvietošanas un iestumšanas stratēģijas sagatavo atmiņu jaunam informācijas blokam;

  Stratēģijas priekšrocības:

  a) ir garantēts, ka atmiņā būs tikai darbam vajadzīgie bloki;
  b) ir minimizēti izdevumi bloka, kuru vajag iestumt, noteikšanai;

- **iestumšana ar apsteidzi** balstās uz tā, ka sistēma paredz programmas nākamos pieprasījumus un jau iepriekš pārraksta nepieciešamo informācijas bloku. Pieņēmums par nākamajiem pieprasījumiem balstās uz programmas konstrukcijas dabu un secinājumiem no procesa iepriekšējās uzvedības.

  Metodes priekšrocības:

a) ja lēmums par bloka izstumšanu ir pieņemts precīzi, tad procesa izpildīšanas laiks tiek samazināts;
b) ja pareizo lēmumu var realizēt bez liekiem izdevumiem, tad procesa izpildīšana tiek paātrināta, nepalēninot citu aktīvu procesu izpildīšanu.

## I/O sistēmas organizēšana

Tradicionāli I/O sistēmu var uzskatīt par OS projektēšanas vissarežģītāko sfēru, kurā grūti izmantot kopējo pieeju un kurā dominē atšķirīgas metodes. Projektējot I/O sistēmu, vajag izveidot I/O iekārtu virtuālo interfeisu, kas ļauj programmētājiem vienkārši nolasīt datus un saglabāt tos, neņemot vērā iekārtas specifiku. Mērķi, kuriem jāatbilst I/O sistēmām:

1) neatkarība no iekārtu rakstzīmju kodiem. Tas nozīmē, ka I/O sistēmai jābūt atbildīgai par koda atpazīšanu un to pārveidošanu standarta formā lietotāju programmā;
2) I/O iekārtu neatkarība. Šeit var uzrādīt divus aspektus:
    a) programmai jābūt neatkarīgai no uzrādītas iekārtas konkrēta veida;
    b) programmai pēc iespējas jābūt neatkarīgai no I/O iekārtas vispār;
3) Efektivitāte. Tā kā I/O operācijas ir sistēmas šaurā vieta, ir vēlams izpildīt tās ātri un pēc iespējas efektīvāk;
4) Unificētas griešanās pie I/O iekārtām organizēšana. Ir vēlams apstrādāt visas iekārtas kaut kādā unificētā veidā, lai sasniegtu vienkāršību un atbrīvotos no kļūdām.

# 11. Lekcija

## Ievad izvades(I/O) sistēma daļas

- Neatkarība no rakstu kodiem
- I/O sistēmas neatkarība
- Efektivitāte
- Unificēta pieeja

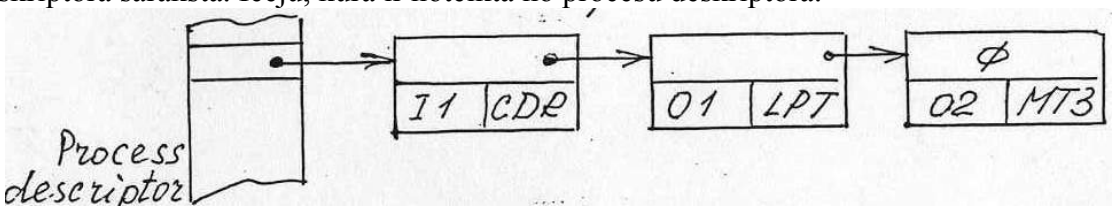
### 1. Neatkarība

No rakstu kodiem nozīmē, ka eksistē visu simbolu unificēta iekšējas attēlošanas forma, kuru sauc par iekšējiem kodiem. Translācijas mehānisms saistīts ar šo kodu pārveidošanu I/O plūsmā. Translācijai izmanto tabulas, retāk nelielas programmas.

### 2. I/O sistēmas neatkarība

Nozīmē, ka programmas sadarbojas nevis ar reālām iekārtām, bet ar virtuālām iekārtām, kurām ir dažādi nosaukumi: plūsmas, faili datu kopas.

I/O plūsmas izmanto programmas, bez atsaukšanas un konkrētu iekārtu. I/O plūsmu piesaistīšana konkrētām iekārtām izpilda OS, izmantojot informāciju, kas atrodama darba aprakstā. Plūsmu un iekārtu ekvivalenci var uzrādīt plūsmu deskriptora sarakstā. Ieeju, kurā ir noteikta no procesu deskriptora.




### 3. Efektivitāte

Nozīmē, ka I/O sistēmu jāvada tā lai iekārtu raksturojumi ātrāk tiktu saistīti ar konkrētām iekārtām nekā ar atbilstošiem atdarinātājiem. Nepieciešamā informācija par iekārtu var saglabāt iekārtas deskriptorā. Tas dod iespēju vienkāršot atdarinātājus.

Deskriptorā tiek saglabāta:

1. iekārtas identifikācija
2. instrukcijas, kuras jāpilda iekārtām
3. Iezīmē karst zīmju translāciju tabulā
4. tekoša stāvokļa noteikšana(aizņemta, brīva….)
5. Pašreizējā lietotāja procesa iezīme uz procesa deskriptoru.

Visus iekārtas deskriptorus var apvienot iekārtu struktūrā uz kuru ir iezīmē no centrālās tabulas.

### 4. Unificēta pieeja

Iekārtu un plūsmu deskriptori ir svarīgi līdzekļi dažādu mērķu sasniegšanai.

Tipisks procesa vaicājums I/O procesa izpildīšanai var izskatīties tā:

DOIO (stream, mode, amount, destination, semaphore)

Radītāju apraksts

Stream- plūsmas numurus

Mode- nosaka operācijas, kura ir pieprasīta
Amount- datu apjums
Destination- adresāts
Semaphore- semafors „request serviced" adrese. Šī semafora signalizē, kad I/O operācija ir pabeigta.
Procedūru DOIO var izmantot vienlaicīgi dažādi procesi
Procedūras funkcijas ir sekojošas

1. Piesaistīt fizisku iekārtu I/O plūsmai
2. Parādīt uzrādīto parametrus un iekārtas raksturojumu atbilstību
3. Inicializēt pieprasīto servisu

Pirmās divas funkcijas var realizēt izmantojot plūsmu deskriptoru sarakstu un iekārtas deskriptoru.

Pēc tam I/O procedūru izpilda vaicājuma parametru asamblēšanu. I/O sistēmas vaicājuma bloks IORB pievieno analoģisko bloku rindai uz konkrētu iekārtu. Iekārtu vaicājumu rinda ir pievienota iekārtas deskriptoram un tiek apkalpota atsevišķi procesu, kuru sauc par iekārtas atdarinātāju. I/O procedūra atdarinātājam, ka viņš tiek iekļauts rindā. Šim mērķim izmanto semaforu „Request pending" kura ir saistīta ar iekārtu un ir iekļauta iekārtas deskriptoru. Kad I/O operācija ir pabeigta iekārtas atdarinātājs paziņo par to lietotāja procesam izmantojot semaforu „request serviced" . Šī semafors var būt inicializēt gan lietotāja procesu, gan ar I/O procedūru kad tiek veidot IORB

Procedūras kopējs apraksts. Procedūras DOIO (parametri) izpilda visas procedūras

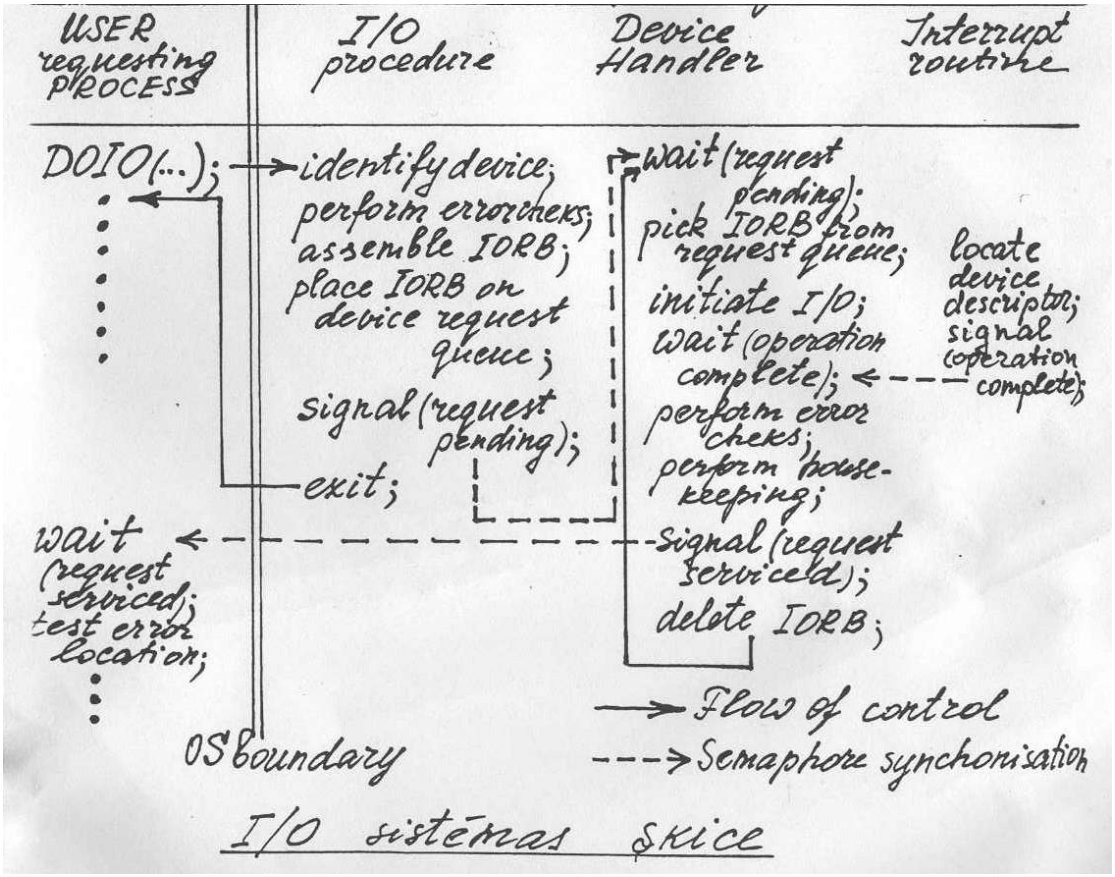
## Iekārtas atdarinātais

Tas ir process, kas ir paredzēts vaicājumu apkalpošanai iekārtu vaicājumu rindā un paziņošanai oriģināla procesā par operācijas pabeigšanu. Katrai iekārtai ir savs atdarinātais, bet par cik visi atdarinātāji ir līdzīgi, tāpēc to var izmantot koplietošanas programmas. Dažādas atšķirības atdarinātāju uzvedībā izceļas no iekārtu raksturojumiem, kurus var iegūt no iekārtas deskriptora. Iekārtas iedala cikliski rindā inicializējot operāciju. Sagaida operācija pabeigšanu un paziņo par to oriģinālām procesam.

Paskaidro I/O operācija

1. Semafora „reques pending" signalizē I/O procedūrai katru reizi, kad vaicājumu rinda ir ievietots IORB, ja rinda ir tukša, tad semafora vērtība ir „0" un iekārtas atdarinātājs ir apstādināts.
2. Atdarinātājs var izņemt IORB no rinda atbilstoši prioritātes algoritmam, bet biežāk ir FIFO
3. Instrukcijas I/O operācijas inicializēšanu var saņemt no iekārtas raksturojumiem, kas tiek saglabāti iekārtas deksktiptora.
4. Semafors „Operation comlete" saņem signālu no pārtraukuma rutīnas, kad pārtraukums ir ģenerēts iekārtā.
5. Kļūdu pārbaude notiek, pēc operācijas pabeigšana ar iekārtas stāvokļu aptauju, ja kļūda tiek konstatēta, tad informācija par to tiek ievietota atbilstošā laukā IORB
6. Rakstzīmju translācija notiek atbilstoši MODE(DOIO), kas ir uzrādīts IORB un translāciju tabulai informācija par kuru atrodas iekārtas deskriptorā.
7. Gadījumā ar izvades operācijām datu saņemšana no avota un rakstzīmju translācija notiek pirms operācijas inicializēšanas.

8. Semafora „Request service” adrese tiek pārsūtīta iekārtas atdarinātājam kā IORB komponente.




I/O sistēma skice var redzēt sinhronizācijas un kontroles plūsmas. Jāatzīmē ka lietotāja process turpinās asinhroni iekārtas atdarinātāja darbā. Šis pieejas nepilnība ir tā kā procesam jābūt atbildīgam par sinhronizāciju ar iekārtu atdarinātājiem. Cita pieeja ir tāda, ka atbildību par sinhronizēšanu var ievietot procedūrā. To var sasniegt ar semafora “request serviced” lokalizēšanu I/O procedūrā ievietojot tajā operācijas “reques pending”

1. gadījumā lietotāja procesam ir iespēja turpināties paralēli ar I/O operāciju, bet pašam jābūt atbildīgam par I/O operācijas pabeigšanu.
2. Atbildība tiek pārcelta no procesa I/O procedūru, bet procesam nosūt iespēju norisināties paralēli I/O procesam

## Buferizācija

Visi paskaidrojumi par I/O procedūru, iekārtas atdarinātāju ir izdarīti pieņēmumā, ka dati pārsūtīšana neizmanto buferizāciju.

Buferizācija ir atmiņas izmantošana atmiņā, datu īslaicīgai glabāšanai, lai realizētu datu apmaiņu starp atmiņu un ārējām ierīcēm. OS organizē ieejas datu pārsūtīšanu ieejas buferī no kura datus saņem lietotāja process. Lietotāja procesam vajadzēs gaidīt tikai ja buferis ir tukšs, šajā gadījumā OS piepildīs buferi un lietotāja process var turpināties. Līdzīgi ir arī ar ieejas datiem. Vēl lielāku efektu var panākt izmantojot ne vienu, bet gan divus buferus. Ja process izdod I/O vaicājumus ar lielu ātrumu, tad paātrināt darbību var ar buferu skaitu palielināšanu 3....N. Vajadzīga vai nē buferizācija ir jānorāda uzdevuma aprakstā, ja vajadzīgi tad jāuzrāda cik buferi ir vajadzīgi. OS realizē atmiņas buferizāciju, kad datu plūsma tiek atvērta uz ieraksta buferu adreses plūsmas deskriptorā. Protams, ka mainās I/O procedūra ar buferizāciju izsaukšanu no lietotāja procesa var izskatīties sekojoši

Pārsūtāmās informācijas izmērs vienmēr ir 1 vienība. Buferizācijas mērķis ir buferizācijas adreses un izmēru visu šo informāciju var dabūt no plūsmu deskriptora saraksta. Šī shēma darbojas tikai uz vienpusējām iekārtam (printeris, klaviatūra u.t.t)