Robert Meolic
meolic@uni-mb.si
zadnja sprememba: 12. 02. 2009

Snov iz drugih predmetov, ki je potrebna za razumevanje predmeta VSO

interno gradivo za predmet VSO, 2008/09

1. LITERATURA

Pri sestavljanju tega gradiva sem uporabljal internet in naslednje vire:

- Abraham Silberschatz, Peter B. Galvin, Greg Gagne: Operating System Concepts, Sixth Edition. John Wiley & Sons, 2002
- Andrej Štrancar: Prosojnice za predmet VSO, 2006/07

2. KOMPONENTE RAČUNALNIŠKEGA SISTEMA

Osnovne komponente vsakega modernega računalniškega sistema so:

- **strojna oprema** – zagotavlja osnovne računske vire (CPE, pomnilnik, V/I naprave),
- **operacijski sistem in sistemska programska oprema** – nadzoruje ter koordinira uporabo strojne opreme za različne uporabniške programe in za različne uporabnike,
- **uporabniški programi** – uporabniki jih uporabljajo pri svojem delu, izvajanje uporabniških programov je glavni namen vsakega računalniškega sistema.

Najpomembnejše **naloge operacijskega sistema**:

- omogočiti izvajanje uporabniških programov,
- narediti računalniški sistem enostaven za uporabo.
- izrabiti strojno opremo na učinkovit način.

3. VRSTE RAČUNALNIŠKIH SISTEMOV

Računalniške sisteme lahko razdelimo v različne skupine.

Enoopravilni (single-tasking) sistemi omogočajo so sočasno izvajanje le enega programa. Če hočemo izvajati več programov, moramo med njimi ročno preklapljati.

Večopravilni (multi-tasking) sistemi dovoljujejo (navidezno) sočasno izvajanje več programov. V resnici računalnik izmenično dodeljuje posameznim programom časovne rezine. Največ časovnih rezin običajno dobiva program, ki deluje v ospredju. Manj rezin dobivajo programi, ki delujejo v ozadju. Najmanj časa je posvečeno programom, ki trenutno ne delajo nič.

Mnogouporabniški (multi-user) sistemi dovoljujejo uporabo istega računalnika, včasih celo istega programa, večim uporabnikom. Taki sistemi pogosto delujejo po principu delitve časa med uporabniki. Za mnogouporabniške sisteme je značilna uporaba navideznega pomnilnika.

Večprocesorski (multi-processor) sistemi temeljijo na uporabi več procesnih enot (CPE), ki so tesno povezane (uporabljajo skupni pomnilnik). Procesne enote so dodeljevanje posameznim programom. Hitrost izvajanja programov na takem sistemu je lahko večja, ni pa nujno. Poveča se lahko tudi zanesljivost, saj odpoved enega procesorja zgolj zmanjša zmogljivost. Pri večprocesorskih sistemih ločimo med **simetričnimi** in **asimetričnimi** modeli. V prvem primeru so vsi procesorji enakovredni, pri asimetričnih modelih pa obstaja glavni procesor, ki nadzoruje delo

ostalih. Asimetrični modeli so bolj pogosto v zelo velikih sistemih, kjer npr. operacijski sistem uporablja en procesor, uporabniška programska oprema pa druge.

Operacijski sistemi za različne skupine sistemov so si med seboj zelo različni. Operacijski sistemi na večopravilnih sistemih so zelo kompleksni, saj morajo npr. skrbeti za učinkovito razvrščanje poslov (kateri program naj se ob nekem času naloži in izvaja), za učinkovito upravljanje s pomnilnikom (kako se izogniti prevelikemu drobljenju - fragmentaciji), za učinkovito zaščito programov (napaka v enem programu ne sme škodovati drugemu programu), za učinkovito komunikacijo med programi ter njihovo sinhronizacijo itd.

4. DELOVANJE RAČUNALNIŠKEGA SISTEMA

Programe na računalniškem sistemu izvaja **procesor**. Programi so za procesor zaporedje strojnih inštrukcij. Izvajanje ene inštrukcije imenujemo inštrukcijski cikel. Inštrukcijski cikel sestavljajo bralni, pisalni in bralno-pisalni cikli, ki jih s skupnim imenom imenujemo strojni cikli. Izvajanje strojnih ciklov traja določeno število urinih impulzov oz. urinih ciklov.

Delovanje procesorja in ostalih delov računalniškega sistema je usklajeno na sinhroni ali asinhroni način. Pri sinhronem načinu napravi uporabljata isti urin takt, pri asinhronem načinu pa vsaka naprava deluje s svojo hitrostjo in se med seboj usklajujeta preko nadzornih signalov.

Gradnik računalniškega sistema, ki je najtesneje povezan s procesorjem, je **delovni oz. glavni pomnilnik (RAM)**. Procesor z delovnim pomnilnikom komunicira preko podatkovnega in naslovnega vodila.

Vhodno/izhodne naprave in procesor delajo vzporedno. Vhodno/izhodne naprave imajo krmilnike, katerih glavna naloga je komunikacija s procesorjem in z delovnim pomnilnikom. Krmilniki vhodno/izhodnih naprav imajo običajno tudi medpomnilnik (buffer). Izmenjava podatkov med medpolnilnikom vhodno/izhodne naprave in glavnim pomnilnikom se običajno izvrši na zahtevo vhodno/izhodne naprave, opravi pa jo procesor. To zahtevo imenujemo **prekinitev**. Prekinitev, ki ima dovolj veliko prioriteto, je odobrena in v tem primeru procesor takoj po koncu tekočega inštrukcijskega cikla začne izvajati ustrezni prekinitveni servisni program. Kje v pomnilniku se ta program nahaja, je določeno s prekinitvenim vektorjem, ki vsebuje naslov prvega ukaza prekinitvenega servisnega programa. Obstaja pa tudi možnost, da vhodno/izhodna naprava brez sodelovanja procesorja sama izmenja podatke z glavnim pomnilnikom, pri čemer se uporabita isto naslovno in podatkovno vodilo, kot ga uporablja procesor, zato se morata vhodno/izhodna naprava in procesor pri takem načinu prenosa med seboj uskladiti. Tak način prenosa podatkov imenujemo **neposredni dostop do pomnilnika (DMA)**.

Posebno poglavje delovanja računalniških sistemov je njihov **zagon (booting)**. Ob zagonu računalniškega sistema se požene program, ki je shranjen v posebnem čipu (lahko je ROM, EEPROM ali pa flash pomnilnik), ki pregleda in inicializira strojno opremo ter sproži nalaganje operacijskega sistema. Tradicionalno to funkcijo opravlja program imenovan **Basic Input-Output System (BIOS)**. Moderni BIOS je precej izpopolnjena različica prvotnega, ki se je pojavil skupaj s PC-ji. Ker pa samo z dopolnjevanjem BIOS-a ne moremo zagotoviti učinkovitosti, ki si je danes želimo, so se pojavile bolj zmogljive alternative, med katerimi je najpomembnejša **Extensible Firmware Interface (EFI)**. Zagon računalniškega sistema se konča tako, da se v RAM naloži določen del nekega diska in se ga izvede. Na tem delu diska se mora nahajati program, ki sproži nalaganje operacijskega sistema. Operacijski sistem ima za potrebe zagona poseben inicializacijski program (initial system file), ki se požene kot prvi program na računalniku. Ta inicializacijski program potem požene še druge dele operacijskega sistema, npr. grafični uporabniški vmesnik.

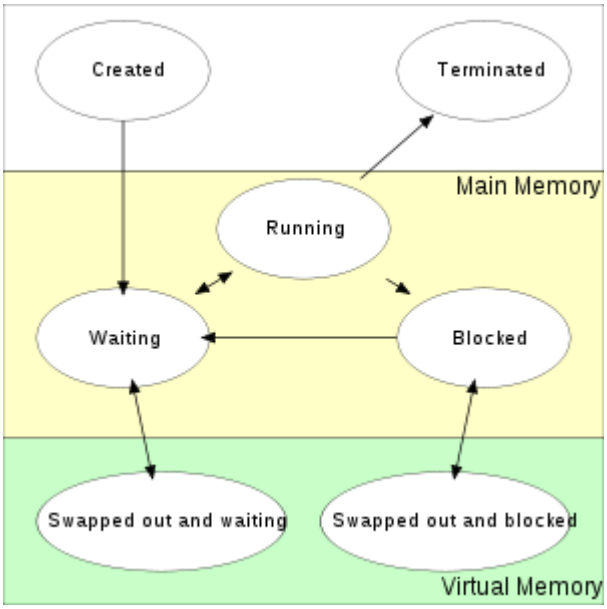
5. PROCESI

Proces (process) je program v izvajanju. Ima pasivni del (programska koda) in aktivni del (trenutne vrednosti spremenljivk). Operacijski sistem ne dela neposredno s programi ampak s procesi. Pomembne naloge operacijskega sistema glede procesov so:

- tvorjenje in uničevanje procesov,
- porazdeljevanje sistemskih virov posameznim procesom,
- zaščita procesov pred napakami v drugih procesih in
- sinhronizacijo procesov ter komunikacija med njimi.

Med izvajanjem vsak proces prehaja med različnimi **stanji**:

- ustvarjen (created), to je začetno stanje, v katerega se nikoli več ne vrne (včasih označen tudi kot nov - new),
- teče (running), če se ravnokar izvajajo njegovi ukazi,
- čaka (waiting), če čaka, da pride na vrsto za izvajanje, (včasih označen tudi kot pripravljen – ready),
- blokiran (blocked), če čaka na nek dogodek (npr. konec V/I aktivnosti, signal), da bi lahko spet stekel (včasih označen tudi kot spi - sleeping),
- pripravljen (ready), če mu manjka le še procesor,
- končan (terminated), če je zaključil izvajanje programa



Aktivni del procesa je shranjen v **nadzornem bloku procesa (process control block, PCB)**, ki vsebuje:

- stanje procesa
- programski števec
- vsebina registrov
- podatki potrebni pri razvrščanju procesa,
- podatki za upravljanje s pomnilnikom
- podatki povezani z uporabljanimi vzhodno/izhodnimi napravami,
- drugi "računovodski" podatki, npr. seznam odprtih datotek.

Operacijski sistem si pri delu s procesi pomaga z **vrstami (queues)**:

- vrsta vseh procesov – vsi trenutno obstoječi procesi,
- vrsta pripravljenih procesov – procesi v delovnem pomnilniku, ki so pripravljeni na izvajaje,
- vrste za vhodno/izhodne naprave – procesi, ki čakajo na posamezno vhodno/izhodno napravo.

Ko preklopimo iz enega procesa v drugega, mora operacijski sistem shraniti stanje starega procesa v njegov PCB in naložiti shranjeno stanje za nov proces. To imenujemo **zamenjava okolja (context switch)**. Čas, ki je potreben za zamenjavo okolja, je izgubljen, saj računalniški sistem med tem opravilom ne dela koristnega dela. Čas potreben za zamenjavo okolja je odvisen od podpore strojne opreme.

Kako se ustvari nov proces? Ustvari ga lahko operacijski sistem ali pa drug proces. V slednjem primeru pravimo, da je nov proces sin obstoječega procesa oz. da je obstoječi proces oče novega procesa. Običajno je, da proces sin od procesa oče podeduje vse sistemske vire in da oče tudi določi njegove začetne podatke. Običajno je tudi, da proces oče po tvorjenju novega procesa nadaljuje svoje izvajanje.

Kako se proces konča? Obstaja več možnosti, npr. da proces konča samega sebe (exit), da proces oče zahteva končanje procesa sina (abort) ali da operacijski sistem konča proces (če npr. ta prekorači dodeljene sistemske vire). Večina operacijskih sistemov ne dovoli, da bi sinovi obstajali, če se proces oče konča, zato se uporablja verižno (kaskadno) zaključevanje procesov.

Vmesnik med procesi in operacijskim sistemom predstavljajo **sistemski klici**. Z njimi procesi zahtevajo neko storitev operacijskega sistema (npr. exit, abort).

6. NITI IN OPRAVILA

Za doseganje večopravilnosti na modernih operacijskih sistemih poleg procesov uporabimo tudi niti (threads) oz. opravila (tasks). Razlika med procesom in nitjo je na različnih operacijskih sistemih različna, v splošnem pa velja, da je zamenjava okolja pri nitih lažja, hitrejša in manj zahtevna kot zamenjava okolja pri procesih.

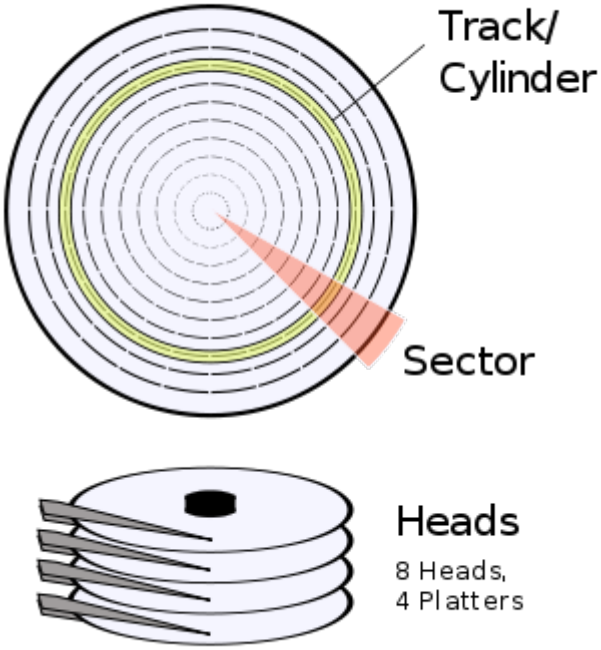
Za pravilno razdelitev izvajanja programa v več niti mora poskrbeti programer oz. prevajalnik - operacijski sistem programa ne zna razdeliti procesa v več niti ampak samo skrbi za njihovo izvajanje.

Večnitni programi se lahko zelo učinkovito izvajajo na večprocesorskih sistemih.

7. DISKI

Diski so tipična naprava za hranjenje podatkov, ki omogoča naključni dostop. To pomeni, da lahko zaporedoma dostopamo do podatkov, ki so fizično shranjeni na oddaljenih mestih diska. Vendar pa imajo diski tudi svoje omejitve med katerimi je najpomembnejša ta, da je najmanjša količina podatkov, ki jo lahko naenkrat preberemo ali zapišemo 512 B. Ta osnovna enota se imenuje **blok (block)** ali pa **sektor (sector)**, saj je včasih ustrezala enemu fizičnemu sektorju na disku.

Bloki morajo biti nekako označeni, da jih lahko naslovimo. Tradicionalno se je uporabljala oznaka, ki je bila sestavljena iz treh delov: oznaka cilindra, oznaka glave in oznaka fizičnega sektorja. Ta način označevanja s kratico imenujemo oznaka CHS (cylinder-head-sector) in je odražala fizično zgradbo diska, kakor je prikazana na spodnji sliki.



S pojavom ATA in SCSI diskov (starejše generacije so se imenovale drugače, npr. MFM diski, RLL diski, …) se je pojavilo neenakomerno zapisovanje podatkov na diske in tako imajo zunanji cilindri, ki so večji, več podatkov od notranjih. Označevanje blokov je pri teh diskih sicer ostalo enako, vendar pa nima več neposredne zveze s fizično razporeditvijo. Modernejši način označevanje se imenuje LBA (logical block addressing), pri katerem bloke po vrsti oštevilčimo od nič naprej, krmilnik diska pa potem poskrbi, da se ta številka preslika v fizično lokacijo.

Da omogočimo več različnih operacijskih sistemov in/ali več različnih datotečnih sistemov na enem disku, vpeljemo **particije (partitions).** Informacija o razdelitvi diska je shranjena na samem disku v posebni tabeli. Ta tabela se nahaja v prvih blokih (torej tistem s številko LBA 0 in naslednjih), njena kopija pa je včasih tudi na zadnjih blokih. Prvi blok na disku imenujemo **Master Boot Record (MBR)**. Tabela, ki tradicionalno opisuje razdelitev particij, se imenuje **Master Partition Table (MPT)** in je v celoti del MBR. Tabela MPT predvideva, da je položaj particij na disku zapisan z oznakami CHS, vendar pa danes tam najdemo številke, ki ne odražajo pravega stanja (tipično tam za začetek in konec particije piše Cylinder 1023, Head 254, Sec 63). Modernejši sistemi danes namesto MPT že uporabljajo **GUID Partition Table (GPT)**, ki se nahaja izven MBR in bo v prihodnosti prevzela vodilno vlogo.

8. OSNOVE DATOTEČNIH SISTEMOV

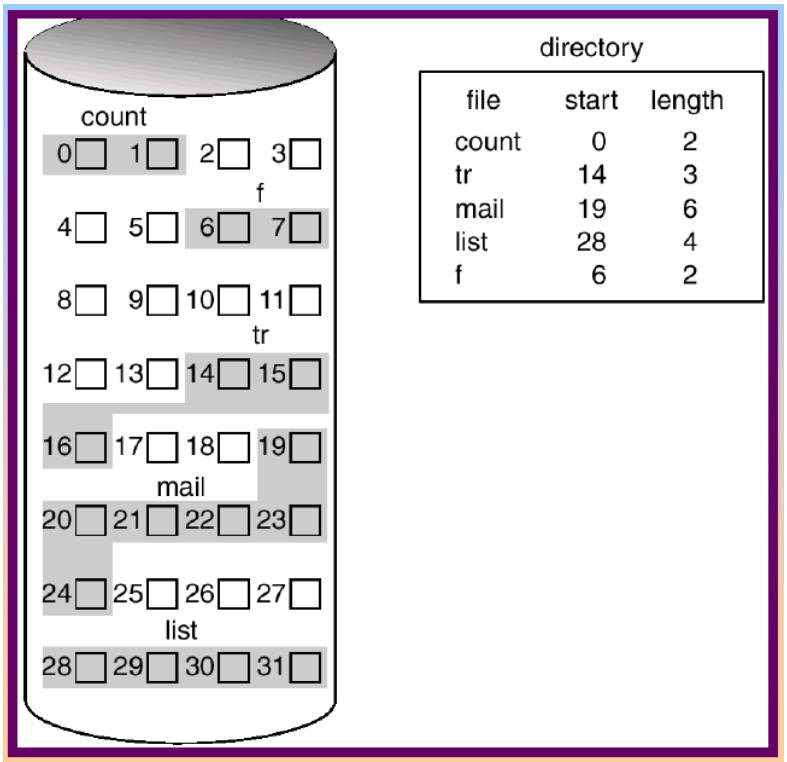
Datotečni sistem je način, kako organizirati podatke na pomnilniških medijih. Osnovna elementa datotečnega sistema sta **datoteka (file)** in **mapa oz. imenik (folder, directory)**. Vsaki datoteki in vsaki mapi pripadajo razni atributi, npr. lastnik, velikost, datum nastanka, zaščita itd.

Osnovna enota, ki ji pripada datotečni sistem, se imenuje **zvezek (volume)**. To je lahko cel disk, en del diska, izmenljivi medij itd. Vsak zvezek je sestavljen iz oštevilčenih koščkov, ki jih imenujemo **segmenti (segments)**. Velikost segmentov je določena s fizičnimi lastnostmi medija, npr. diski naenkrat preberejo oz. zapišejo bloke velikosti 512 B, na CD-jih pa so ti bloki enaki 2048 B. Način, kako zapišemo zaporedno številko segmenta, določa maksimalno število segmentov v enem zvezku, npr. če bi uporabili 16 bitne številke bi lahko imeli le 65.536 segmentov.

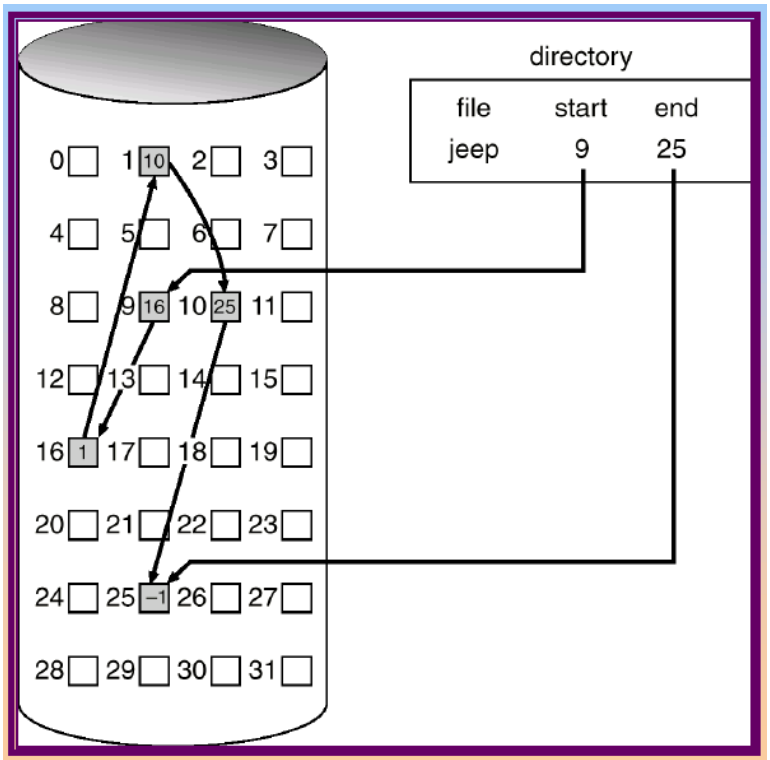
Datoteke so sestavljene iz enega ali več zaporednih ali pa nezaporednih segmentov. Imeniki so lahko izvedeni kot preprost **linearen seznam** (linear list) imen datotek in ustreznih kazalcev na bloke podatkov ali pa kot **razpršene tabele** (hash table).

Način, kako se segmenti na disku dodeljujejo datotekam, imenujemo dodeljevanje prostora (space allocation). Poznamo:

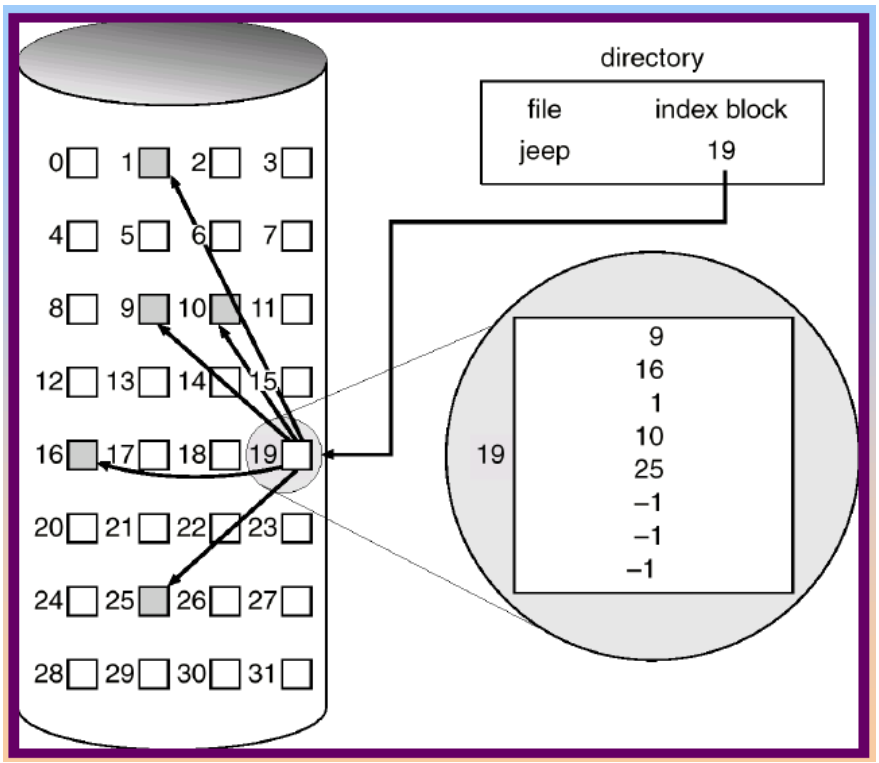
- **zvezno dodeljevanje** (contiguous allocation)- vsaka datoteka zaseda množico zaporednih segmentov na disku, ta način je enostaven za uporabo, ker za vsako datoteko potrebujemo le številko začetnega segmenta in dolžino (število segmentov), slabosti pa sta neučinkovita izraba prostora in dejstvo, da datotek v splošnem ni mogoče enostavno povečevati,



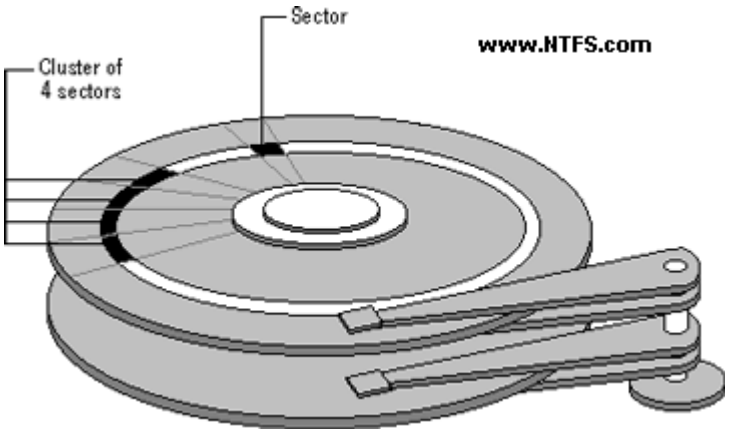
- **povezano dodeljevanje** (linked allocation) - vsaka datoteka je povezan seznam segmentov na disku, pri čemer so segmenti ene datoteke po disku poljubno razmetani, ta način je prav tako enostaven za uporabo, saj potrebujemo samo številko začetnega segmenta in posebno oznako, ki pomeni konec datoteke, povezano dodeljevanje je tudi precej učinkovito glede izrabe prostora, slabost pa je, da ne moremo naključno dostopati do poljubnega segmenta neke datoteke,



- **indeksirano dodeljevanje** - indekse segmentov, ki sestavljajo datoteko, združimo v indeksnem polju, tudi indeksirano dodeljevanje je učinkovito glede izrabe prostora, omogoča pa tudi neposredni dostop do poljubnega segmenta datoteke, slabost je, da del prostora izgubimo zaradi indeksnih polj, če ima datoteka več segmentov, kot jih lahko naštejemo v enem indeksnem polju, uvedemo večnivojsko strukturo,



- **parcelirano dodeljevanje** (extent-based alloaction) - uporablja se v kombinaciji z ostalimi načini dodeljevanja, ideja je v tem, da prostor datotekam dodeljujemo v kosih, ki obsegajo več segmentov. Te večje enote imenujemo bloki (blocks) oz. grozdi (clusters).



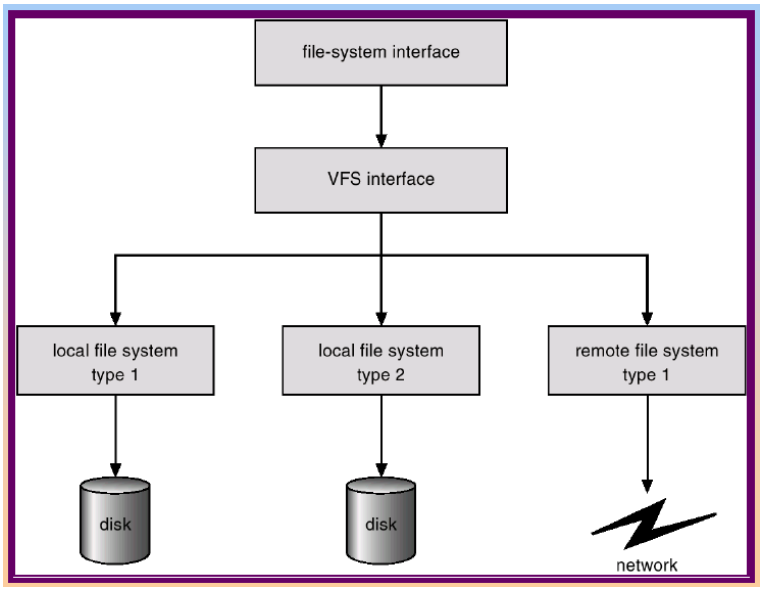
Pri **dodeljevanju prostora** datotekam moramo imeti evidenco o **prostih segmentih** (oz. blokih ali grozdih). Uporabimo lahko **bitni vektor**, v katerem vsak bit ustreza enemu segmentu. Bitni vektor mora biti shranjen na disku. Med delovanjem se uporablja kopija bitnega vektorja v delovnem pomnilniku. Poskrbeti je potrebno, da segment, ki je v delovnem pomnilniku označen kot zaseden, na disku nikoli ni označen kot prost. Zato je zaporedje operacij vedno naslednje:

1.postavi bit[i] = 1 na disku,
2. dodeli segment[i],
3. Postavi bit[i] = 1 v pomnilniku,

Drugi način evidence prostega prostora je **povezan seznam prostih segmentov**.

Datotečni sistemi z dnevnikom obravnavajo vsak poseg kot transakcijo, ki se zabeleži v dnevnik. Transakcije se uvrstijo v dnevnik preden se izvedejo. Ko je transakcija izvršena, se zbriše iz dnevnika. V primeru, da se sistem zruši, se tako ohrani informacija o tem, katere transakcije so bile izvršene in katere še čakajo na izvršitev.

Navidezni datotečni sistem (VFS) je dodatni sloj med uporabnikom in fizičnim datotečnim sistemom. VFS omogoča poenoten način dostopa preko funkcij združenih v poseben API.



S pomočjo VFS lahko uporabnik enotno uporablja datotečni sistem, ki sestoji iz več fizičnih datotečnih sistemov, na eni ali na več fizičnih napravah. To je uporabno iz več razlogov:

- različni sistemi lahko podpirajo različno uporabo.
- poveča se zanesljivost
- možno je izboljšati učinkovitost z nastavitvijo različnih parametrov.
- program ne more zasesti vsega prostora z eno veliko datoteko,
- hitrejše arhiviranje in obnavljanje particij ter večja preglednost arhiva.