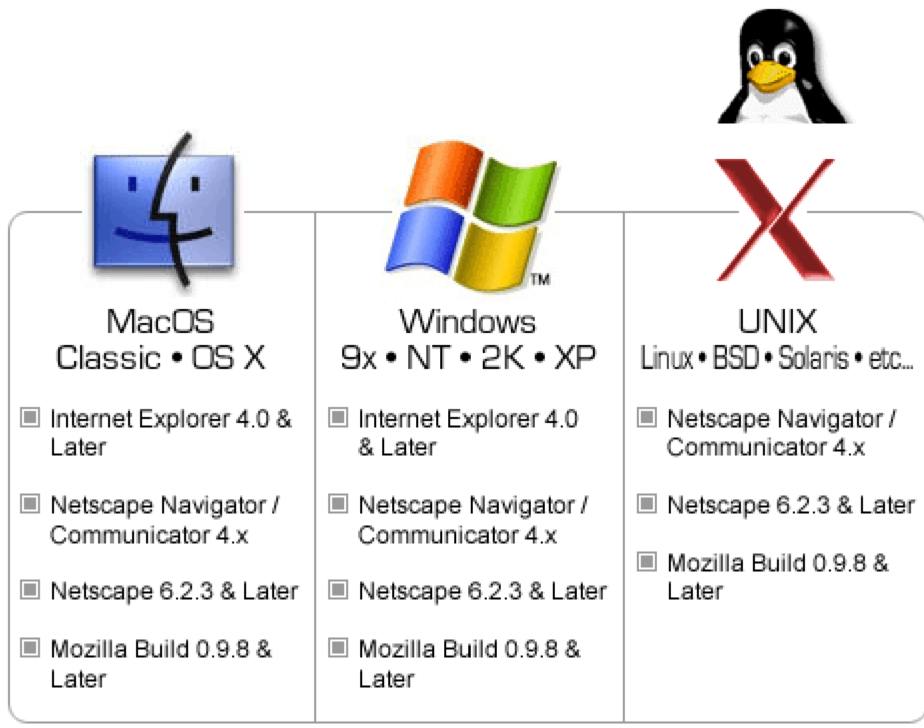
Tehniški šolski center Nova Gorica
Višja strokovna šola

Uvod v operacijske sisteme

gradivo za predmet OPS
višja strokovna šola, smer informatika



© Erik Kompara

Kazalo vsebine

1. Uvod....7
1.1. Kaj počne operacijski sistem?....7
1.1.1. Pogled s strani uporabnika....8
1.1.2. Pogled s strani strojne opreme....8
1.1.3. Cilji operacijskega sistema....9
1.1.4. Sistemi z dodeljevanjem časa – večopravilni sistemi....9
1.2. Večprocesorski sistemi....10
1.3. Porazdeljeni sistemi....11
1.3.1. Odjemalec-strežnik sistemi....11
1.3.2. Sistem enakih računalnikov....11
1.4. Grozdni sistemi (Clustered systems)....12
1.5. Realno časovni sistemi (Real time systems)....12
1.6. Ročni sistemi....12
1.7. Migracija konceptov operacijskih sistemov....12
1.8. Vprašanja za ponavljanje....13
2. Struktura računalniških sistemov....14
2.1. Organizacija računalniškega sistema....14
2.1.1. Delovanje računalniškega sistema....14
2.1.2. Shranjevalne strukture....15
2.1.3. V/I strukture....16
2.1.3.1. V/I prekinitve....16
2.1.3.2. DMA strukture....17
2.1.4. Arhitekture računalniških sistemov....17
2.1.4.1. Enoprocesorski sistemi....17
2.1.4.2. Večprocesorski sistemi....18
2.1.4.3. Grozdni sistemi....18
2.1.5. Strukture operacijskih sistemov....19
2.2. Operacije operacijskega sistema....20
2.2.1. Dvojni režim delovanja....21
2.2.2. Časovnik....22
2.3. Upravljanje procesov....22
2.4. Upravljanje pomnilnika....23
2.5. Upravljanje shranjevanja....23
2.5.1. Upravljanje z datotečnim sistemom....23
2.5.2. Upravljanje z masovnimi napravami za shranjevanje....24
2.5.3. Upravljanje hitrega pomnjenja (caching)....24
2.6. Varnost in zaščita....25
2.7. Porazdeljeni sistemi....26
2.8. Sistemi s posebnim namenom....26
2.8.1. Realno časovni vgrajeni sistemi....26
2.8.2. Ročni sistemi....26
2.9. Računalniška okolja....27

2.9.1. Tradicionalno računalniško okolje........ 27
2.9.2. Okolje odjemalec-strežnik........ 27
2.9.3. Okolje enakovrednih sistemov........ 29
2.10. Vprašanja za ponavljanje........ 30
3. Strukture operacijskih sistemov........ 31
3.1. Sistemske komponente........ 31
3.1.1. Upravljanje procesov........ 31
3.1.2. Upravljanje glavnega pomnilnika........ 32
3.1.3. Upravljanje datotečnega sistema........ 32
3.1.4. Upravljanje V/I sistema........ 32
3.1.5. Upravljanje s sekundarnim pomnilnikom........ 33
3.1.6. Omrežje........ 33
3.1.7. Sistem zaščite........ 33
3.1.8. Sistem ukaznega tolmača........ 34
3.2. Servisi operacijskega sistema........ 34
3.3. Uporabniški vmesnik do operacijskega sistema........ 35
3.3.1. Ukazni tolmač (CLI)........ 35
3.3.2. Grafični uporabniški vmesnik (GLI)........ 36
3.4. Sistemski klici........ 36
3.4.1. Sistemski programi........ 38
3.5. Vprašanja za ponavljanje........ 38
4. Upravljanje procesov........ 40
4.1. Procesi........ 40
4.1.1. Stanja procesa........ 41
4.1.2. Kontrolni blok procesa (Process Control Block – PCB)........ 41
4.1.3. Razvrščanje procesov........ 42
4.1.3.1. Razvrščevalna vrsta........ 43
4.1.3.2. Razvrščevalnik........ 43
4.1.3.3. Menjava okolja (Context switch)........ 45
4.1.4. Operacije nad procesi........ 45
4.1.4.1. Ustvarjanje procesov........ 45
4.1.4.2. Končevanje procesov........ 45
4.1.5. Sodelovanje procesov........ 47
4.1.6. Medprocesno komuniciranje........ 48
4.1.6.1. Sistem prenosa sporočil........ 48
4.1.6.2. Imenovanje........ 48
4.1.6.2.1. Direktno komuniciranje........ 49
4.1.6.2.2. Indirektno komuniciranje........ 49
4.1.6.3. Primer: Windows XP........ 49
4.1.7. Komunikacija odjemalec-strežnik........ 50
4.1.7.1. Vtiči (sockets)........ 50
4.2. Niti........ 51
4.2.1. Uporabniške niti in niti jedra........ 51
4.2.2. Večnitni modeli........ 52
4.2.2.1. Več na ena model........ 52
4.2.2.2. Ena na ena model........ 52
4.2.2.3. Več na več model........ 52

4.3. Vprašanja za ponavljanje....... 53
5. Razvrščanje na procesorju....... 54
5.1. Osnovni koncepti....... 54
5.1.1. V/I delovni cikel procesorja....... 54
5.1.2. CPE razvrščevalnik....... 56
5.1.2.1. Razvrščanje s ali brez prekinjanja ((non)preemptive scheduling)....... 56
5.1.2.2. Dodeljevalnik (dispatcher)....... 56
5.1.3. Kriteriji razvrščanja....... 57
5.1.4. Algoritmi razvrščanja na procesorju....... 57
5.1.4.1. "Prvi pride, prvi melje" algoritem razvrščanja (First-Come, First-Served – FCFS)....... 58
5.1.4.2. "Najkrajše opravilo najprej" algoritem razvrščanja (Shortest-Job-First – SJF)....... 59
5.1.4.3. Prioritetno razvrščanje....... 61
5.1.4.4. Round Robin (RR) razvrščanje....... 61
5.1.4.5. Več-nivojska vrsta razvrščanja....... 64
5.1.4.6. Primer Windows XP....... 65
5.1.4.7. Primer Linux....... 65
5.2. Vprašanja za ponavljanje....... 66
5.3. Sinhronizacija procesov....... 67
5.3.1. Ozadje problema - nekonsistentnost skupnih podatkov....... 67
5.3.2. Problem kritične sekcije....... 68
5.3.2.1. Algoritem 1....... 69
5.3.2.2. Algoritem 2....... 70
5.3.2.3. Algoritem 3....... 71
5.3.3. Sinhronizacija preko strojne opreme....... 71
5.3.3.1. Onemogočanje prekinitev (interrupt disabling)....... 71
5.3.3.2. Strojni ukaz TestAndSet....... 72
5.3.4. Semaforji....... 72
5.3.4.1. Uporaba semaforjev....... 73
5.3.4.2. Implementacija semaforjev....... 73
5.3.5. Smrtni objemi....... 75
5.3.6. Binarni semaforji....... 75
5.3.7. Klasični problemi sinhronizacije....... 76
5.3.7.1. Problem proizvajalec – porabnik (Bounded-Buffer problem)....... 76
5.3.7.2. Problem branja-pisanja (Readers-Writers Problem)....... 77
5.3.7.3. Problem lačnih filozofov (Dining-Philosophers Problem)....... 78
5.3.8. Vprašanja za ponavljanje....... 79
6. Upravljanje shranjevanja....... 80
6.1. Upravljanje pomnilnika....... 80
6.1.1. Ozadje problema....... 80
6.1.2. Osnovna strojna oprema....... 81
6.1.3. Dodeljevanje absolutnih naslovov....... 82
6.1.3.1. Logični naslovni prostor – fizični naslovni prostor....... 83
6.1.3.2. Dinamično nalaganje in povezovanje....... 83
6.1.3.3. Začasno izločanje (swapping)....... 84
6.1.4. Nedeljivo dodeljevanje procesa v pomnilnik....... 85
6.1.4.1. Zaščita pomnilniškega prostora....... 85
6.1.4.2. Naslavljanje pomnilnika....... 86
6.1.5. Drobljenje (fragmentation)....... 87
6.1.6. Stranjenje (paging)....... 87
6.1.6.1. Osnovna metoda....... 88

6.1.6.2. Zaščita ... 90
6.1.6.3. Strani v skupni rabi ... 91
6.1.7. Segmentacija ... 91
6.1.7.1. Osnovna metoda ... 91
6.1.7.2. Strojna oprema ... 92
6.2. Vprašanja in naloge za ponavljanje ... 93
6.3. Virtualni pomnilnik ... 94
6.3.1. Ozadje ... 94
6.3.2. Stranjenje na zahtevo (demand paging) ... 95
6.3.2.1. Osnovni koncept ... 95
6.3.3. Kreiranje procesov ... 97
6.3.3.1. Kopija ob zapisu (Copy-on-Write) ... 97
6.3.3.2. Dodeljevanje datotek v fizičnem pomnilniku (Memory-Maped Files) ... 97
6.3.4. Vprašanja za ponavljanje ... 98
6.4. Vmesnik datotečnega sistema ... 99
6.4.1. Koncept datoteke ... 99
6.4.1.1. Lastnosti datoteke ... 99
6.4.1.2. Operacije nad datoteko ... 100
6.4.1.3. Tipi datotek ... 100
6.4.1.4. Struktura datoteke ... 101
6.4.1.5. Metode dostopa ... 102
6.4.1.5.1. Zaporeden dostop ... 102
6.4.1.5.2. Direktni dostop ... 102
6.4.2. Struktura direktorija ... 102
6.4.2.1. Eno-nivojski direktorij ... 103
6.4.2.2. Dvo-nivojski direktorij ... 104
6.4.2.3. Drevesna struktura direktorija ... 104
6.4.2.4. Acikličen diagram direktorijev ... 105
6.4.2.5. Splošen diagram direktorijev ... 106
6.4.3. Priklop datotečnega sistema ... 106
6.4.4. Skupna raba datotek ... 107
6.4.4.1. Več uporabnikov ... 107
6.4.4.2. Oddaljeni datotečni sistemi ... 107
6.4.4.2.1. Model odjemalec – strežnik ... 108
6.4.4.2.2. Porazdeljeni informacijski sistemi ... 108
6.4.5. Zaščita ... 108
6.4.5.1. Tipi dostopa ... 109
6.4.5.2. Kontrola dostopa ... 109
6.4.6. Vprašanja za ponavljanje ... 111
6.5. Implementacija datotečnega sistema ... 112
6.5.1. Struktura datotečnega sistema ... 112
6.5.2. Implementacija datotečnega sistema ... 113
6.5.2.1. Razdelki in priklapljanje razdelkov ... 114
6.5.2.2. Virtualni datotečni sistem ... 114
6.5.3. Implementacija direktorija ... 115
6.5.3.1. Linearni seznam ... 115
6.5.3.2. Hash seznam ... 115
6.5.4. Metode dodeljevanja prostora ... 116
6.5.4.1. Sosedno dodeljevanje prostora (contiguous allocation) ... 116
6.5.4.2. Povezano dodeljevanje prostora (linked allocation) ... 117
6.5.4.3. Indeksno dodeljevanje prostora ... 118
6.5.5. Upravljanje s prostim prostorom ... 119

6.5.5.1. Bit vektor........ 119
6.5.5.2. Povezan seznam........ 120
6.5.6. Vprašanja za ponavljanje........ 121
6.6. Strukture za masovno shranjevanje........ 122
6.6.1. Struktura diska........ 122
6.6.2. Razvrščanje na disku........ 123
6.6.2.1. Prvi pride prvi melje(FCFS) razvrščanje........ 123
6.6.2.2. Najkrajši dostopni čas najprej (SSTF) razvrščanje........ 124
6.6.2.3. SCAN razvrščanje........ 125
6.6.2.4. Krožni SCAN razvrščanje........ 125
6.6.2.5. LOOK razvrščanje........ 126
6.6.3. Upravljanje z diskom........ 126
6.6.3.1. Formatiranje diska........ 126
6.6.3.2. Zagonski blok (boot blok)........ 127
6.6.4. Vprašanja za ponavljanje........ 128
7. Vhodno / Izhodni sistemi (V/I)........ 129
7.1. Splošno........ 129
7.1.1. Vhodno / Izhodna strojna oprema........ 129
7.1.2. Prekinitve........ 131
7.1.2.1. Direkten dostop do pomnilnika (DMA)........ 132
7.1.2.2. Aplikacijski V/I vmesnik........ 133
7.1.2.3. V/I podsistem jedra........ 134
7.2. Vprašanja za ponavljanje........ 135
8. Viri in literatura........ 136

1. Uvod

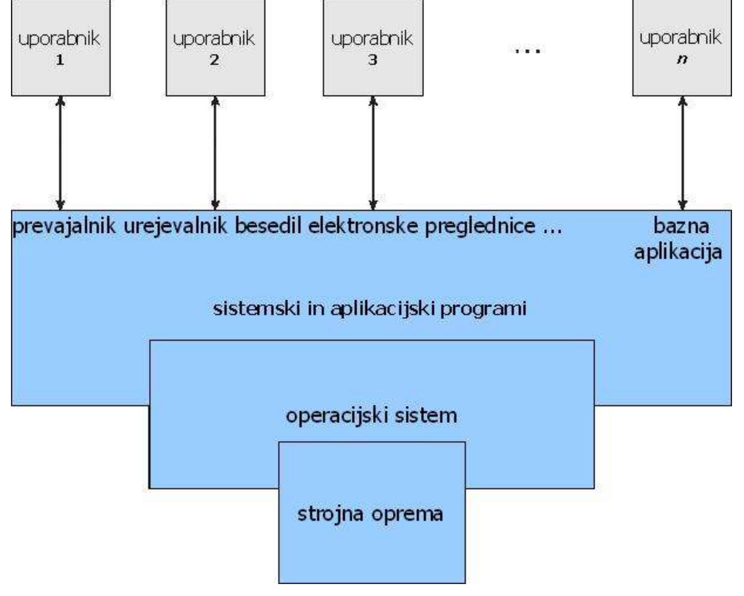
Operacijski sistem je **program**, ki:
- upravlja s strojno opremo računalnika,
- deluje kot osnova in podpora za delovanje aplikacij
- se obnaša kot posrednik med uporabnikom računalnika in strojno opremo računalnika.

Operacijski sistemi se razlikujejo v tem, kako opravijo naštete naloge.
Kot primer lahko povemo, da so na primer operacijski sistemi za **velike računalnike** - "mainframe" narejeni tako, da čimbolj efektivno izkoristijo strojno opremo sistema, na drugi strani pa so operacijski sistemi, ki ustvarjajo okolje, v katerem bo uporabnik čim lažje in udobneje rešil želeno nalogo (**ročni računalniki** (dlančniki, mobilni telefoni, ...)). Vmes med izkoriščenostjo strojne opreme in udobnostjo uporabnika pa so **osebni računalniki**, kjer si operacijski sistem prizadeva, da bi čimbolj izkoristil strojno opremo računalnika ob čimbolj enostavni in udobni uporabi za uporabnika.

Tako so na eni strani operacijski sistemi narejeni tako, da so prijazni do uporabnika - "user frendly, na drugi strani morajo operacijski sistemi čimbolj efektivno izkoriščati vire in tudi taki, ki so kombinacija udobja za uporabnika in efektivnosti izkoriščanja virov.

1.1. Kaj počne operacijski sistem?

Operacijski sistem je pomemben del vsakega računalniškega sistema. Računalniški sistem lahko razdelimo v 4 komponente:
- **strojna oprema**
- **operacijski sistem**
- **aplikacije**
- **uporabniki**



- **Strojna oprema –** procesor, pomnilnik, V/I naprave, zagotavljajo osnovne računalniške vire za sistem
- **Programske aplikacije -** urejevalnik besedil, elektronske preglednice, prevajalniki, spletni brskalniki, ..., ki definirajo kako strojno opremo uporabiti ob tem, da uporabnik opravi nalogo.
- Operacijski sistem **kontrolira in koordinira uporabo strojne opreme**, ki jo uporabljajo sistemski in uporabniški programi, katere poganjajo različni uporabniki.

Operacijski sistem je podoben Vladi države. Sam sebi je nekoristen, zagotavlja pa urejeno okolje znotraj katerega ostali programi delajo koristno delo.

Operacijski sistem lahko gledamo iz dveh zornih kotov:
- pogled s strani uporabnika
- pogled s strani sistema.

1.1.1. Pogled s strani uporabnika

Pogled s strani uporabnika je odvisen od vmesnika, ki ga uporabnik uporablja.
1. Zamislimo si uporabnika, ki sedi pred osebnim računalnikom (monitor, tipkovnica, miška in sistemska enota). Tak sistem je narejen za enega uporabnika, ki si prisvoji vso strojno opremo, da lahko opravlja svoje delo ali igra igrico. V tem primeru je operacijski sistem narejen tako, da je **uporabniku** prijazen in učinkovit. V tem primeru ni važno, da večina strojne opreme počiva oziroma se komaj nekajkrat odzove na uporabnikove (glede na računalniški takt) počasne V/I zahteve. Pojem **efektivnega ali učinkovitega** izkoriščanja strojnih virov je v tem primeru zanemarjen.
2. Zamislimo si bančnega uslužbenca, ki sedi pred terminalom, ki je povezan v večji sistem - "Mainframe". Ostali bančni uslužbenci tudi sedijo pred terminalom in so povezani na isti večji sistem. Uporabniki si lahko preko sistema izmenjujejo informacije. V tem pimeru je operacijski sistem narejen tako, da čimbolj učinkovito izkoristi strojno opremo – zagotovi, da so čas procesorja, pomnilnik in vhodno/izhodne naprave uporabljene efektivno in da vsak uporabnik koristi svoj del strojne opreme.
3. Zamislimo si konstruktorja, ki dela na grafični **delovni postaji**, ki je povezana v omrežje z drugimi konstruktorji in s **strežnikom**. Konstruktorji dajejo svojo strojno opremo v skupno rabo in tudi sami izkoriščajo omrežne storitve kot so dostop do strežnika, tiskalniškega strežnika, ...Tak operacijski sistem mora biti kompromis med prijazno uporabo in učinkovito izkoriščenostjo strojne opreme.
4. Zadnje čase narašča priljubljenost in uporaba ročnih računalnikov. Uporablja ga en uporabnik, zato je operacijski sistem enouporabniški s poudarkom na **uporabnost** in **varčnost**.
5. Nekateri sistemi nimajo uporabniškega pogleda. To so vgrajeni sistemi v domačem gospodinjstvu, avtomobilu, ..., ki imajo komajda kakšno tipko za vnos ali svetlobni indikator, ki kaže stanje. Večinoma so v te naprave vgrajeni operacijski sistemi, ki delujejo brez uporabnika.

Lahko bi sklenili pogled z razmišljanjem, da operacijski sistem poizkuša gledano s strani uporabnika in uporabnosti prikriti abstraktnost delovanja strojne opreme.

1.1.2. Pogled s strani strojne opreme

Operacijski sistem je program, ki je v tesni povezavi s strojno opremo računalnika. Računalniški sistem ima veliko virov, tako strojnih kot programskih, ki pripomorejo k rešitvi naloge (čas procesorja, pomnilniški prostor, shranjevalni prostor, V/I naprave, ...).
- Operacijski sistem **nadzoruje in usmerja uporabo strojne opreme**,
- Operacijski sistem **dodeljuje vire (resource allocation)**.
- V primeru, da dve aplikaciji istočasno zahtevata določen vir, mora operacijski sistem **nastalo konfliktno situacijo rešiti pravično in efektivno**.
- **Jedro** (**kernel**) je edini program v operacijskem sistemu, ki deluje ves čas ko je računalnik vključen, vse ostalo, kar dobimo od proizvajalca operacijski sistemov so sistemski in aplikacijski programi.

1.1.3. Cilji operacijskega sistema

Primarni cilj operacijskega sistema je, da **naredi strojno opremo računalnika uporabniku koristno**. Operacijski sistemi obstajajo, ker je lažje reševati naloge z njimi, kot brez njih. To je jasno, ko govorimo o operacijskem sistemu za mikroračunalnike (osebne računalnike).
Operacijski sistemi za **velike računalnike** pa imajo glavni cilj **učinkovito delovanje** računalniškega sistema – torej učinkovitost. Koristnost in učinkovitost sta si včasih kontradiktorna, saj pomeni na primer uvedba grafičnega vmesnika, ki prispeva k prijaznosti sistema tudi obremenitev pomnilnika in s tem slabšo učinkovitost sistema.
Da si lažje predstavljamo kaj so operacijski sistemi in kaj počnejo preučimo, kako so se operacijski sistemi zgodovinsko razvijali.

1.1.4. Sistemi z dodeljevanjem časa – večopravilni sistemi

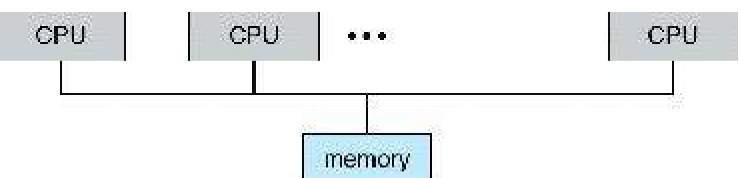
Potreba po hitrem odzivu na odpravo napak in direktnem posredovanju programerja je rodilo idejo o **večopravilnem sistemu**, kjer ima vsak uporabnik možnost interakcije z računalnikom. Večopravilni sistemi so pripeljali do komuniciranja programerja z računalnikom preko zaslona in tipkovnice. Ob tem je bila tudi obvezna programska oprema kot je **urejevalnik teksta**, **razhroščevalnik** (debugger) in uporaba datotek, v katerih ima programer svoje programe in podatke. Pojavil se je pojem **organizacije datotek** – **datotečni sistem**.
Večopravilni sistem lahko gledamo tudi kot podaljšek miltiprogramirnega sistema, saj poleg obdelave več opravil naenkrat omogoča tudi interaktivno delo z računalnikom več uporabnikom hkrati.
Ker večopravilni sistem preklaplja zelo hitro od enega uporabnika do drugega in so odzivi uporabnika zelo počasni glede na hitrost sistema, se vsakemu uporabniku zdi, da se računalnik ves čas ukvarja z njim. Vsak uporabnik v večopravilnem sistemu ima vsaj en program v pomnilniku, ki se izvaja. Temu programu pravimo **proces**. Ko se proces izvaja se izvaja zelo kratek čas. Za tem se proces konča ali čaka na V/I operacijo. V/I operacija je lahko **interaktivna** – program pošlje na monitor zahtevo, uporabnik pa preko tipkovnice ali miške zahtevi ugodi.
Take operacije so zelo počasne (človek tipka na tipkovnico največ 7 znakov/s) v primerjavi z hitrostjo sistema, zato so take operacije normalne za človeka, zelo počasne pa za računalniški sistem. Da nebi procesor čakal, operacijski sistem preklopi na program drugega uporabnika in ga začne izvajati.
Prvi resni večopravilni operacijski sistem je bil narejen leta 1962 in se je imenoval **CTSS** (Compatible Time Sharing System).
Po uspehu CTSS sistemov in združitvi velikih podjetij, ki so se ukvarjala z računalniško tehnologijo (MIT, Bell Labs, General Electric) se je leta 1970 pojavil večopravilni sistem, ki naj bi deloval po načelu distribucije električne energije – ko potrebuješ elektriko vtakneš napravo v vtič in na razpolago ti je toliko energijo kolikor jo potrebuješ.
Večopravilni sistem naj bi nudil računalniške storitve več sto uporabnikom naenkrat in pokril celo mesto (Boston). Sistem so imenovali **MULTICS** (MULTiplexed Information and Computing Service).
Kot zanimivost omenimo, da so nekatera velika podjetja (General Motors, Ford, U.S. National Security Agency, ...) ugasnila njihove MULTICS sisteme šele leta 1990.

1.2. Večprocesorski sistemi

Večina sistemov je enoprocesorskih, kar pomeni, da ima sistem samo en procesor. Večprocesorski ali vzporedni sistemi so vedno bolj pomembni. Taki sistemi imajo več kot en procesor v tesni povezavi. Večprocesorski sistem ima rti glavne prednosti:

1. **Povečan odzivni čas**. S povečanjem procesorjev upamo, da bo sistem opravil več v krajšem času. Faktor povečanega odziva sistema z N procesorji je manjši kot N, ker mora operacijski sistem poleg izvajanja opravil skrbeti, da procesorji med seboj pravilno delujejo.
2. **Ekonomska upravičenost**. Večprocesorski sistem prihrani denar, ker si več procesorjev deli isto strojno opremo (naprave za shranjevanje podatkov, napajanje, ...). Če več programov obdeluje neke podatke je ceneje shraniti podatke na en disk in imeti več procesorjev, ki jih obdeluje, kot imeti več računalnikov z lokalnim diskom in več kopijami podatkov.
3. **Zanesljivost**. Če so naloge procesorjev pravilno razporejene na procesorje, napaka na enem procesorju ne povzroči zamrznitve celotnega sistema. Če imamo 10 procesorjev in se eden pokvari si naloge razdeli ostalih 9. Tako je sistem le 10% počasnejši. Temu pravimo tudi **mila degradacija sistema** (graceful degradation), sistemom pa **odporni na napake** (fault tolerant).

Običajni večprocesorski sistemi uporabljajo **simetrično večprocesiranje (SMP)**, pri katerem vsak procesor vzdržuje identično kopijo operacijskega sistema in kopije komunicirajo med seboj ko je potrebno.



Nekateri sistemi
uporabljajo **asimetrično večprocesiranje**, pri katerem je vsakemu procesorju dodeljena določena naloga. Glavni procesor kontrolira sistem, ostali procesorji čakajo na ukaze glavnega procesorja ali imajo predefinirane naloge. Tak sistem definira gospodar-služabnik zvezo. Glavni procesor dodeljuje naloge podprocesorjem.

Današnji operacijski sistemi kot so Windows NT, Solaris, Digital Unix, OS/2 in Linux podpirajo podporo za simetrično večprocesiranje.

1.3. Porazdeljeni sistemi

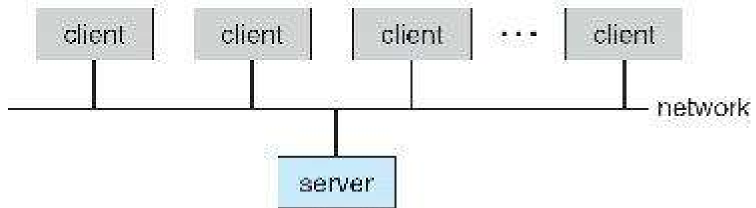
Porazdeljeni sistemi so odvisni od omrežja, ki jih povezuje. Porazdeljeni sistemi so zmožni komunicirati in s tem deliti računska opravila. Omrežje je odvisno od protokola, ki ga uporablja, razdalje in prenosnimi mediji med sistemi. Omrežja se delijo glede na razdaljo med sistemi.

- LAN (Local Area Network) omrežje je značilno za porazdeljene sisteme, ki se nahajajo znotraj ene sobe, nadstropja ali stavbe.
- WAN (Wide Area Network) omrežje je značilno za porazdeljene sisteme, ki se nahajajo med stavbami, mesti ali celo državami. Kot primer vzemimo podjetje, ki ima širom sveta svoje poslovne enota in mora enote primerno povezati.
- MAN (Metropolitan Area Network) omrežje je značilno za porazdeljene sisteme, ki povezujejo zgradbe znotraj mesta.
- SAN (Small Area Network) - BlueTooth naprave komunicirajo preko kratke razdalje in v osnovi ustvarijo majhno omrežje.

1.3.1. Odjemalec-strežnik sistemi

Ko so osebni računalniki postajali cenejši, hitrejši in močnejši so se načrtovalci vedno bolj oddaljevali od centraliziranih sistemov. Terminale počasi zamenjujejo osebni računalniki. Vlogo centralnih sistemov prevzemajo strežniki, ki strežejo zahtevam odjemalcev, ki zamenjujejo terminale. Strežniške sisteme lahko kategoriziramo na aplikacijske strežnike in datotečne strežnike:

- aplikacijski strežnik, kjer se zahteva odjemalca izvrši, rezultat zahteve se pošlje nazaj odjemalcu. Kot primer vzemimo, da na aplikacijskem strežniku teče spletni strežnik. Uporabnik na odjemalcu napiše internetni naslov neke spletne strani. Internetni naslov se prenese do ustreznega spletnega strežnika, kjer se zahtevana spletna stran pošlje nazaj in odjemalec vidi zahtevano spletno stran.
- datotečni strežnik, ki priskrbi datotečni sistem, kjer odjemalec lahko shrani, spreminja, bere in briše datoteke.



1.3.2. Sistem enakih računalnikov

Prvi operacijski sistemi so bili narejeni za osebno rabo, osamljeni in nepovezani. Z uvedbo Internetnih storitev (elektronska pošta, FTP, WWW, ...) sredi 90 let je mrežna povezljivost bila nepogrešljiva komponenta operacijskega sistema. Skoraj vsi današnji operacijski sistemi imajo internetni brskalnik za prikazovanje hipertekstnih dokumentov, protokole (TCP/IP, PPP, ...), ki omogočajo sistemu, da lahko komunicira z drugimi sistemi v omrežju.
Govorimo o **mrežnem operacijski sistemu**, ki je operacijski sistem, ki omogoča skupno rabo datotek, komunikacijske sheme, kjer si lahko različni procesi na različnih računalnikih

izmenjujejo sporočila. Računalnik, ki poganja mrežni operacijski sistem deluje avtonomno, neodvisno od drugih računalnikov v omrežju.

1.4. Grozdni sistemi (Clustered systems)

Enako kot večprocesorski sistemi, grozdni sistemi zberejo skupaj več procesorjev, da opravijo neko nalogo hitreje. Grozdni sistemi se razlikujejo od večprocesorskih po tem, da so grozdni sistemi sestavljeni iz več kot enega posameznega sistema. Grozdni sistemi si delijo naprave za shranjevaje in so tesno povezani preko LAN omrežja. Sistemi skrbijo za visoko razpoložljivost. Vsak posamezen sistem nadzira ostale in v primeru izpada nekega sistema v grozdu prevzame njegove naloge drugi sistem. Uporabnik izpada enega sistema ne zazna in opravlja svoje naloge naprej.

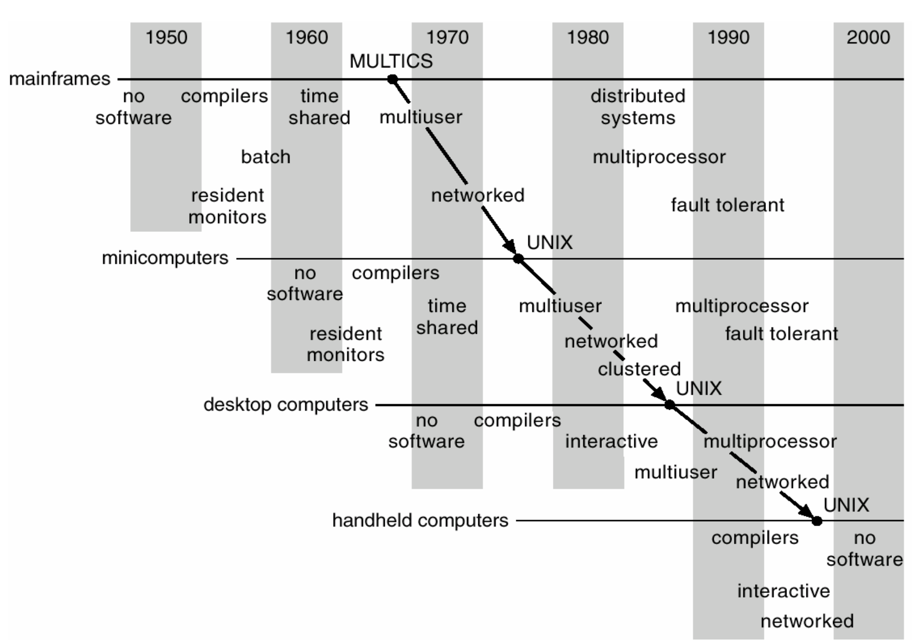
1.5. Realno časovni sistemi (Real time systems)

To je oblika namenskega operacijskega sistema, kjer se zaradi strogih časovnih omejitev zahteva usklajena obdelava nalog ali pretoka podatkov. To so sistemi, ki kontrolirajo znanstvene poizkuse, medicinske obsevalni sistemi, industrijski kontrolni sistemi, bolj konkretno recimo sistem za elektronsko vbrizgavanje goriva .

1.6. Ročni sistemi

Sem spadajo osebni digitalni asistenti, kot so dlančniki, mobilni telefoni, ki se lahko povežejo v omrežje, na primer Internet. Načrtovalci ročnih sistemov se spopadajo z mnogimi pastmi, ki jih prinašajo 12 cm x 7 cm velike napravice. Taki sistemi imajo malo spomina, majhno procesorsko moč, majhne tipke in zaslon. Operacijski sitem mora v takem primeru pomnilnik zelo efektivno uporabljati.

1.7. Migracija konceptov operacijskih sistemov



1.8. Vprašanja za ponavljanje

1. Naštej tri glavne namene operacijskega sistema?

2. Naštej 4 komponente, ki so potrebne za izvajanje programa na sistemu.

3. Kaj je glavna prednost multiprogramiranja?

4. Kaj je bil glavni vzrok uveljavitve večopravilnega sistema?

5. Definiraj glavne značilnosti operacijskih sistemov, ki so:
   - paketni (še ni operacijski sistem)
   - interaktivni
   - večopravilni
   - realno-časovni
   - mrežni
   - vzporedni ali večprocesorski
   - porazdeljeni
   - grozdni
   - ročni

6. Kakšne so 3 glavne prednosti multiprocesiranja?

7. Razloži razliko med simetričnim in asimetričnim multiprocesiranjem.

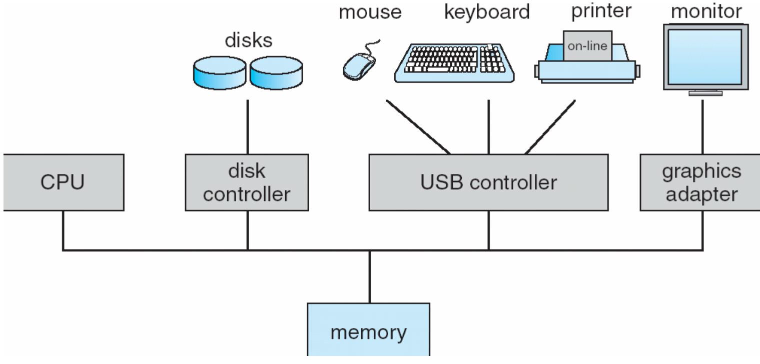
8. Kakšne so glavne razlike med operacijskim sistemom za velike računalnike in operacijskim sistemom za osebne računalnike?

2. Struktura računalniških sistemov

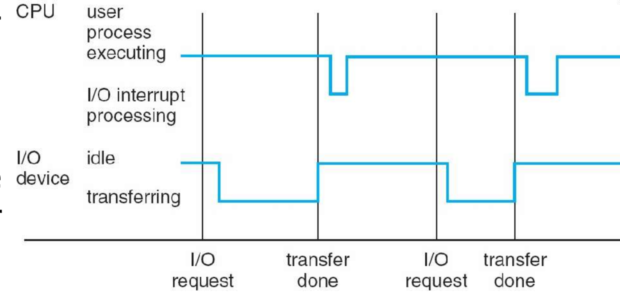
2.1. Organizacija računalniškega sistema

Najprej si bomo pogledali strukturo računalniškega sistema in nato operacije operacijskega sistema.

2.1.1. Delovanje računalniškega sistema



Moderen računalniški sistem se sestoji iz enega ali več procesorjev in kontrolerjev naprav, ki so preko **skupnega vodila** povezani s skupnim pomnilnikom.
Računalnik za njegov zagon (ko ga vžgemo ali ponovno zaženemo) potrebuje začetni program. Program mora biti enostaven in je ponavadi shranjen v pomnilniku vrste ROM (Read Only Memory) ali EEPROM (Electrical Eraseable Programabile Read Only Memory), ki se nahaja na matični plošči. Osnovna naloga začetnega programa je ta, da preveri in inicializira procesorske registre procesorja, kotrolerjev in določi vsebino delovnega pomnilnika. Poleg tega mora vedeti kako prenesti jedro (kernel) operacijskega sistema iz diska v delovni pomnilnik. Operacijski sistem nato zažene prvi proces – pri Unix sistemih proces z imenom init. Proces nato čaka na dogodke. Znak, da se bo zgodil nek dogodek se signalizira preko strojnih ali programskih prekinitev.
Strojna oprema sporoča procesorju, da bi rada pozornost preko prekinitve.
Uporabniški ali sistemski program pa sporoča procesorju, da bi rad pozornost preko **sistemskih klicev** (system calls).
Vsi moderni operacijski sistemi upoštevajo prekinitveno delovanje.
Za vsako vrsto prekinitve požene operacijski sistem ločen segment programske kode, ki ji pravimo **prekinitvena servisna rutina** (interrupt service routine). Ko je procesor prekinjen, takoj ustavi tekoče delo in prebere določeno - fiksno lokacijo v glavnem pomnilniku, kjer je začetni naslov prekinitvene rutine. Izvajati začne prekinitveno rutino in ko jo konča, procesor začne zopet tekoče delo tam, kjer ga je končal pred prekinitvijo.
Na sliki vidimo uporabniški proces, ki da zahtevo strojni opremi za prenos podatkov iz strojne opreme. Ko strojna oprema izvede prenos podatkov, izvede prekinitev in procesor pospravi trenutno stanje njegovih registrov na varno in začne izvajati prekinitveno rutino (na primer prenos podatkov iz medpomnilnika naprave v glavni pomnilnik). Ko konča se vrednosti registrov zopet postavijo in procesor dela naprej. Itd...



2.1.2. Shranjevalne strukture

Programi, ki se izvajajo morajo biti shranjeni v delovnem pomnilniku – RAM-u (Random Access Memory). Delovni pomnilnik je večji pomnilnik, ki lahko shrani na milijone bajtov. Do podatkov v delovnem pomnilniku lahko pride procesor direktno. To omogoča polprevodniška tehnologija imenovana **dinamično naključno naslavljanje pomnilnika** (Dynamic Random Access Memory ali DRAM). DRAM tvori polje pomnilniških besed, kjer ima vsaka beseda svoj naslov. Interakcija s procesorjem se zgodi z dvema instrukcijama – naloži (load) in shrani (store). **Instrukcija** "store" shrani register v procesorju na lokacijo v pomnilniku. instrukcija "load" naloži iz pomnilniške lokacije v register v procesorju. Tipičen cikel, kjer se navodilo izvede najprej pobere iz pomnilnika navodilo in ga naloži v register z navodili (instruction register) v procesorju. Po tem, ko se navodilo v registru prevede lahko zahteva tudi operande, ki so zopet shranjeni v pomnilniku. Ko se navodilo izvede do konca, se rezultat zopet shrani v pomnilnik.

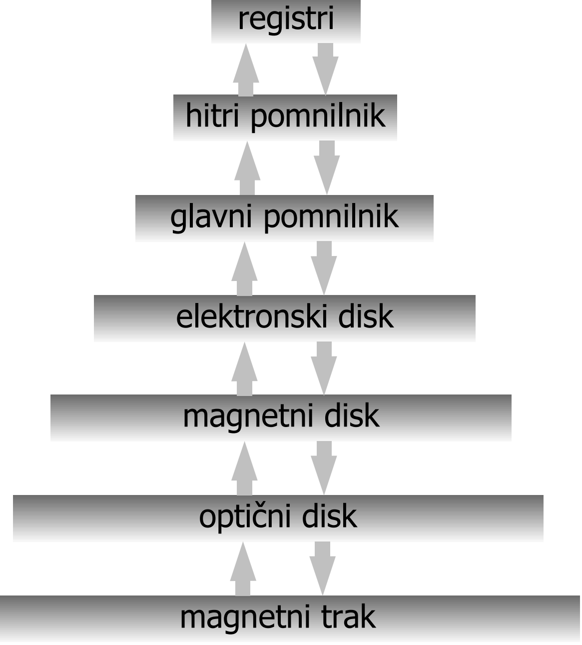
Vedeti moramo, da pomnilnik pri teh operacijah vidi samo tok naslovov, kjer bodo podatki shranjeni, ne pa kako so naslovi generirani (programski števec, indeksni števec, ...) in kakšna je vsebina na teh naslovih (navodila ali podatki). Ne zanima nas kako so naslovi generirani preko programa, ampak tok naslovov, ki jih generirajo programi.

Idealno bi bilo, da bili programi in podatki shranjeni na enem mestu trajno – v glavnem pomnilniku. Vendar to ni mogoče iz dveh razlogov.

- glavni pomnilnik **je premajhen**, da bi shranil vse mogoče programe, ali vsaj kar potrebujemo
- glavni pomnilnik **ne hrani vsebine trajno**, in jo izgubi, ko nima več napajalne energije.

Tako večina računalnikov vsebuje **sekundarno shranjevanje podatkov**. Glavna naloga teh naprav je **trajno shranjevanje velike količine podatkov**. Najbolj razširjen predstavnik sekundarnih virov za shranjevanje podatkov je **magnetni disk**. Vsi sistemski in uporabniški programi so shranjeni na magnetnem disku predenj jih operacijski sistem naloži v glavni pomnilnik.

Poleg omenjenih shranjevalnih struktur (registri, glavni pomnilnik, magnetni disk) obstajajo še druge naprave za shranjevanje podatkov kot so hitri pomnilnik (cache), optični disk, magnetni trakovi, ... Glavne razlike med pomnilnimi napravami so v kapaciteti shranjevanja, hitrosti dostopa, trajnosti zapisa in nenazadnje tudi ceni.



Hierarhija naprav za shranjevanje

Različne tipe shranjevalnih struktur lahko organiziramo v hierarhijo glede na hitrost in ceno. Višji nivoji so dražji a hitrejši. Ko se spuščamo cena/bit pada, dostopni čas pa narašča. To je logično, saj če bi obstajale hitre in cenene naprave, nebi bilo smiselno uporabljati počasne naprave.

Zgornji trije tipi shranjevanja podatkov temeljijo na polprevodniški tehnologiji pomnjenja.

Poleg cene in hitrosti dostopa delimo shranjevalne strukture tudi na začasno ali trajno zapisljive. Začasno zapisljive izgubijo vsebino, če nimajo napajalne energije. Trajno zapisljive ohranijo zapis, tudi, če napajalna energija zmanjka. Spodje tri enote so zmožne shranjevati podatke trajno.
Elektronski disk ima lahko oba načina zapisa podatkov. V primeru trajnega zapisa vsebujejo elektronski diski magnetni disk, kjer se v primeru prekinitve osnovnega napajanja vključi pomožno napajanje, ki shrani vsebino začasnega spomina na magnetni disk. V primeru vrnitve osnovnega napajanja se vsebina iz magnetnega diska prenese v hitrejši elektronski disk.

2.1.3. V/I strukture

Shranjevalne naprave so le en tip V/I naprav. Veliko kode operacijskega sistema je posvečena upravljanju V/I naprav.
Sistem za splošne namene se sestoji iz procesorja in kontrolerjev naprav, ki so med seboj povezani s skupnim vodilom. Kontrolerji naprav vsebujejo lokalne registre in kopico registrov posebnega pomena (disk kontroler vsebuje registre za štetje sektorjev, naslovov diska, pomnilnika, ...). Velikost lokalnega registra je na primer pri kontrolerju za disk enaka ali za določen faktor večja od najmanjšega naslovljenega delca na disku – sektorja, ki je običajno 512B.

2.1.3.1. V/I prekinitve

Da se V/I operacija začne mora procesor naložiti v registre kotrolerja določene vrednosti. Kontroler prebere vsebino registrov, kjer je določeno kaj naj kontroler počne z napravo. Če dobi zahtevo za branje iz naprave bo kontroler začel prenos podatkov iz naprave v lokalne registre. Ko prenese vse podatke v lokalne registre obvesti procesor preko prekinitve.
Poznamo dve vrsti V/I operacij, ki jih opravi uporabniški proces, ko zahteva V/I operacijo.
Sinhrona V/I operacija (na sliki a): ko se V/I operacija začne, se kontrola procesorja prenese na kontroler. Uporabniški proces v tem primeru ne bo nadaljeval izvajanja dokler se V/I operacija ne zaključi, oz. procesor prevzame nazaj izvajanje uporabniškega procesa.
Asinhrona V/I operacija (na sliki b): ko se V/I operacija začne, se kontrola procesorja prenese na kontroler in v tem primeru takoj nazaj na uporabniški proces. V tem primeru uporabniški proces ne čaka da se V/I operacija zaključi.
Čakanje V/I operacije, da konča lahko prikažemo na dva načina. Nekateri računalniki imajo posebno navodilo (wait), ki ustavi delovanje procesorja dokler ne pride naslednja prekinitev. Računalniki, ki nimajo wait instrukcije pa rešijo z zanko (zanka: jmp zanka).
Zavedati se moramo, da imamo več V/I naprav in postavlja se vprašanje kako upravljati z več prekinitvami v enakem trenutku.
V ta namen operacijski sistem vzdržuje tabelo s stanji V/I naprav. Vsak zapis v tabeli govori o tipu naprave, naslovu in trenutnem statusu (zaseden, čaka, na voljo). Če je naprava zasedena z zahtevo po napravi, tip zahteve, in še drugi parametri se shranijo v zapis v tabeli za določeno napravo. Če ostali procesi dajo zahtevo po isti napravi, operacijski sistem vzdržuje čakalno vrsto – spisek čakalnih zahtev za vsako V/I napravo.
Ko V/I naprava zahteva pozornost preko prekinitve, operacijski sistem najprej pogleda katera naprava je prožila prekinitev. Nato gre pogledat v tabelo kakšen status ima naprava in ga ustrezno popravi glede na to zakaj je prišlo do prekinitve. Za večino naprav prekinitev pomeni zaključek zahteve po napravi. Če so v čakalni vrsti še druge zahteve po

tej napravi operacijski sistem začne izvajati naslednjo zahtevo.

2.1.3.2. DMA strukture

Ponovimo procesiranje , kjer proces čaka uporabnika, da vtipka vrstico preko terminala. Ko se prvi znak prenese v računalnik, zaporedna vrata na katera je terminal priključen pošljejo prekinitev CPE. Ko zahteva za prekinitev prispe iz terminala in je ravno takrat CPE na sredini izvajanja instrukcije, CPE počaka, da se izvajanje instrukcije konča. Naslov prekinjene instrukcije se shrani in kontrola se prenese na prekinitveno rutino za terminal. Prekinitvena rutina shrani vse registre v CPE, ki jih bo potrebovala, ko bo CPE nadaljeval prekinjeno delo. Prekinitvena rutina nato shrani znak iz naprave v register. Poleg tega mora popraviti še kazalce in števce, da se naslednji znak shranil v naslednjo lokacijo v registru. Prekinitvena rutina nato postavi zastavico v pomnilniku, ki druge dele operacijskega sistema opozarja, da je v sistem prispel nov znak. Drugi deli so odgovorni, da se znaki prenesejo v program, ki je zahteval vhodno operacijo (glej zaščita strojne opreme). Nato prekinitvena rutina prenese vse shranjene registre nazaj v CPE in preda kontrolo CPE, ki nadaljuje prekinjeno delo.

Če so znaki vneseni preko 9600 baud hitrega terminala, porabijo serijska vrata za prenos znaka naprej 1 ms ali 1000 µs. Dobro napisana prekinitvena rutina prenese znak v register v 2 µs. Tako ostane CPE, da ostalih 988 µs porabi za drugo delo.

Hitrejše naprave kot so trdi diski, mrežna komunikacija, ...lahko prenašajo informacije s hitrostjo , ki je blizu hitrosti pomnilnika. Če CPE potrebuje 2 µs za odziv na prekinitev in prekinitve pridejo vsake 4 µs, pomeni, da CPE nima časa izvajati ostalo delo.

Za rešitev tega problema uporabimo pri hitrih V/I napravah (npr. video podatki) **DMA** (Direct Memory Access). Po tem ko so postavljeni registri, kazalci za V/I napravo, kontroler V/I naprave prenese celoten blok podatkov iz svojih registrov direktno v pomnilnik, brez intervencije CPE. Tako pride do samo ene prekinitve na prenesen blok podatkov, namesto ene prekinitve na prenesen bajt.

2.1.4. Arhitekture računalniških sistemov

2.1.4.1. Enoprocesorski sistemi

Na enoprocesorskih sistemih obstaja **en glavni procesor**, ki je sposoben izvajati splošno-namenske instrukcije, vključujoč instrukcije uporabniških programov.

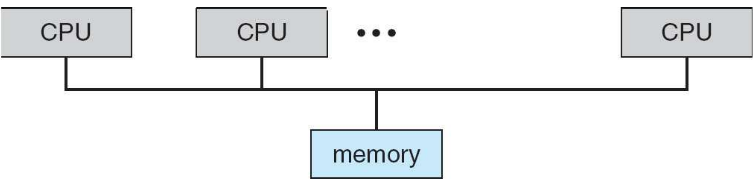
Skoraj vsi sistemi imajo tudi procesorje s **posebnim namenom**, ki so vgrajeni v naprave sistema. Ti procesorji opravljajo točno določeno nalogo (kontrolerji naprav kot so disk, tipkovnica, grafična kartica) ali na velikih računalnikih (mainframe) procesor skrbi za hitre V/I operacije med uporabnikom in sistemom. Ti procesorji imajo omejen nabor instrukcij in ne morejo izvajati uporabniških programov. Procesorje usmerja operacijski sistem, tako da jim podaja informacije kaj je njihova naslednja naloga, ki jo morajo opraviti. Procesor kontrolerja magnetnega diska sprejema zaporedje ukazov od procesorja in vzdržuje vrsto za zahteve ter upravlja razvrščevalni algoritem zahtev. Tipkovnica vsebuje procesor, ki pretvarja pritisnjene tipke v kode, ki se prenesejo v procesor. Operacijski sistem ne mora direktno komunicirati s temi procesorji, zato je delo teh procesorjev avtonomno.

2.1.4.2. Večprocesorski sistemi

Večina sistemov je enoprocesorskih, kar pomeni, da ima sistem samo en glavni procesor. Večprocesorski ali **vzporedni** sistemi so vedno bolj pomembni. Taki sistemi imajo več kot en procesor v tesni povezavi. Večprocesorski sistem ima tri glavne prednosti:

1. **Povečan odzivni čas**. S povečanjem procesorjev upamo, da bo sistem opravil več v krajšem času. Faktor povečanega odziva sistema z N procesorji je manjši kot N, ker mora operacijski sistem poleg izvajanja opravil skrbeti, da procesorji med seboj pravilno delujejo.
2. **Ekonomska upravičenost**. Večprocesorski sistem prihrani denar, ker si več procesorjev deli isto strojno opremo (naprave za shranjevanje podatkov, napajanje, ...). Če več programov obdeluje neke podatke je ceneje shraniti podatke na en disk in imeti več procesorjev, ki jih obdeluje, kot imeti več računalnikov z lokalnim diskom in več kopijami podatkov.
3. **Zanesljivost**. Če so naloge procesorjev pravilno razporejene na procesorje, napaka na enem procesorju ne povzroči zamrznitve celotnega sistema. Če imamo 10 procesorjev in se eden pokvari si naloge razdeli ostalih 9. Tako je sistem le 10% počasnejši. Temu pravimo tudi **mila degradacija sistema** (graceful degradation), sistemom pa **odporni na napake** (fault tolerant).

Običajni večprocesorski sistemi uporabljajo **simetrično multiprocesiranje (SMP)**, pri katerem vsak procesor opravlja vsako nalogo znotraj operacijskega sistema.



Nekateri sistemi uporabljajo **asimetrično multiprocesiranje**, pri katerem je vsakemu procesorju dodeljena določena naloga. Glavni procesor kontrolira sistem, ostali procesorji čakajo na ukaze glavnega procesorja ali imajo predefinirane naloge. Tak sistem definira gospodar-služabnik zvezo. Glavni procesor dodeljuje naloge podprocesorjem.

Današnji operacijski sistemi kot so Windows NT, Solaris, Digital Unix, OS/2 in Linux podpirajo podporo za simetrično multiprocesiranje.

2.1.4.3. Grozdni sistemi

Enako kot večprocesorski sistemi, grozdni sistemi zberejo skupaj več procesorjev, da opravijo neko nalogo hitreje. Grozdni sistemi se razlikujejo od večprocesorskih po tem, da so grozdni sistemi sestavljeni iz več kot enega posameznega sistema. Grozdni sistemi si delijo naprave za shranjevaje in so tesno povezani preko LAN omrežja. Sistemi skrbijo za visoko razpoložljivost. Vsak posamezen sistem nadzira ostale in v primeru izpada nekega sistema v grozdu prevzame njegove naloge drugi sistem. Uporabnik izpada enega sistema ne zazna in opravlja svoje naloge naprej.

2.1.5. Strukture operacijskih sistemov

Eden najpomembnejših pogledov na operacijske sisteme je zmožnost **multiprogramiranja**.

Ideja, ki je razvila multiprogramiranje je bila v tem, da se pomnilnik razdeli na razdelke. V vsakem razdelku leži določeno opravilo. Operacijski sistem naloži v procesor enega od poslov v pomnilniku. Posel zaradi njegove narave recimo čaka na V/I operacijo (vnos znaka iz tipkovnice). Pri starejših načinih reševanja opravil bi v takem primeru procesor počival. Pri multiprogramiranju operacijski sistem prekopi na drugi posel in ga začne izvajati. Ko drugi posel želi čakati, operacijski sistem zopet preklopi na tretji posel in ga začne izvajati, itn. V primeru, da je prvo opravilo dobilo kar je čakalo, zahteva nazaj procesor, ga dobi in prvo opravilo se izvaja naprej.

Multiprogramiranje torej omogoča, da procesor ne počiva in je tako maksimalno izkoriščen. Ko se je neko opravilo končalo je operacijski sistem izpraznil razdelek v pomnilniku in na prazno mesto naložil novega iz bazena opravil (npr. diska).

Tako delovanje lahko predstavimo v vsakdanjem življenju. Za primer vzemimo odvetnika, ki po opravljenih pogovorih s stranko ne čaka na sojenje na sodišču, ampak vzame drugo stranko in z njo zopet opravi potrebne pogovore, itn.

Multiprogramiranje je prvi primer, kjer mora operacijski sistem odločati namesto človeka.

Če je več poslov pripravljenih na prenos v pomnilnik in v pomnilniku ni dovolj prostora mora operacijski sistem izbirati med njimi. Tako pridemo do nove funkcije operacijskega sistema – **razvrščanje poslov** (job scheduling).

Ko operacijski sistem izbere posel iz bazena, ga naloži v pomnilnik, kjer se bo izvajal. Ko imamo več poslov v pomnilniku, je potrebno pomnilnik pravilno upravljati. Tako pridemo do nove funkcije operacijskega sistema - **upravljanje pomnilnika** (memory management).

Če je več poslov pripravljenih za izvajanje v istem trenutku mora operacijski sistem izbirati med njimi. Tako pridemo do nove funkcije operacijskega sistema - **razvrščanje na procesorju** (CPU scheduling).

Ko več poslov teče istočasno moramo zagotoviti, da ne vplivajo eden na drugega – **zaščita poslov**.

Ko posli tečejo, zahtevajo tudi medsebojno komunikacijo – **komunikacija in sinhronizacija poslov**.

Čeprav je bil napredek v strojni opremi in programju, so se pojavljale tudi slabosti. Ti sistemi so bili v osnovi še vedno paketni sistemi. Neko opravilo je lahko trajalo tudi več ur. Opravilo je v primeru napake v programu zastalo. Programer pa je v takem primeru zapravil tudi pol dneva, da je popravil program. Govorimo o temu, da so programerji pogrešali čase prvih sistemov, kjer je bilo možno posredovati preko komandne plošče, ko se je opravilo ustavilo. Problem je torej bil v **možnosti posredovanja programerja** v primeru napak (debugging) in s tem problem majhne storilnosti.

Potreba po hitrem odzivu na odpravo napak in direktnem posredovanju programerja je rodilo idejo o **večopravilnem sistemu**, kjer ima vsak uporabnik možnost interakcije z računalnikom. Večopravilni sistemi so pripeljali do komuniciranja programerja z

računalnikom preko zaslona in tipkovnice. Ob tem je bila tudi obvezna programska oprema kot je **urejevalnik teksta**, **razhroščevalnik** (debugger) in uporaba datotek, v katerih ima programer svoje programe in podatke. Pojavil se je pojem **organizacije datotek – datotečni sistem**.
Večopravilni sistem lahko gledamo tudi kot podaljšek miltiprogramirnega sistema, saj poleg obdelave več opravil naenkrat omogoča tudi interaktivno delo z računalnikom več uporabnikom hkrati.
Ker večopravilni sistem preklaplja zelo hitro od enega uporabnika do drugega in so odzivi uporabnika zelo počasni glede na hitrost sistema, se vsakemu uporabniku zdi, da se računalnik ves čas ukvarja z njim. Vsak uporabnik v večopravilnem sistemu ima vsaj en program v pomnilniku, ki se izvaja. Temu programu pravimo **proces**. Ko se proces izvaja se izvaja zelo kratek čas. Za tem se proces konča ali čaka na V/I operacijo. V/I operacija je lahko **interaktivna** – program pošlje na monitor zahtevo, uporabnik pa preko tipkovnice ali miške zahtevi ugodi.
Take operacije so zelo počasne (človek tipka na tipkovnico največ 7 znakov/s) v primerjavi z hitrostjo sistema, zato so take operacije normalne za človeka, zelo počasne pa za računalniški sistem. Da nebi procesor čakal, operacijski sistem preklopi na program drugega uporabnika in ga začne izvajati.
Prvi resni večopravilni operacijski sistem je bil narejen leta 1962 in se je imenoval **CTSS** (Compatible Time Sharing System).
Po uspehu CTSS sistemov in združitvi velikih podjetij, ki so se ukvarjala z računalniško tehnologijo (MIT, Bell Labs, General Electric) se je leta 1970 pojavil večopravilni sistem, ki naj bi deloval po načelu distribucije električne energije – ko potrebuješ elektriko vtakneš napravo v vtič in na razpolago ti je toliko energijo kolikor jo potrebuješ.
Večopravilni sistem naj bi nudil računalniške storitve več sto uporabnikom naenkrat in pokril celo mesto (Boston). Sistem so imenovali **MULTICS** (MULTiplexed Information and Computing Service).
Kot zanimivost omenimo, da so nekatera velika podjetja (General Motors, Ford, U.S. National Security Agency, ...) ugasnila njihove MULTICS sisteme šele leta 1990.

2.2. Operacije operacijskega sistema

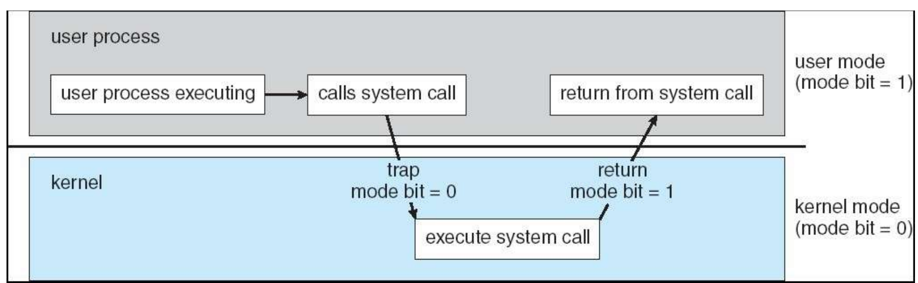
Moderni operacijski sistemi so vodeni preko prekinitev.
Če ni procesa, ki bi se izvajal, ni V/I operacije, ki bi ji bilo treba ustreči in ni uporabnika, ki bi mu bilo treba odgovoriti, operacijski sistem čaka na dogodek. Dogodki so vedno signalizirani preko **prekinitev** (interrupt) ali **pasti** (trap, exception).
Past je prekinitev, ki je programske narave, kadar pride do napake (npr. naslavljanje zaščitenega pomnilniškega prostora ali deljenje z 0) ali če program pošlje zahtevo za izvajanje posebnega servisa OS (npr. branje z V/I naprave).
Za vsako vrsto prekinitve požene operacijski sistem ločen segment programske kode, ki določi kaj naj se izvede ob prekinitvi. Tej kodi pravimo **prekinitvena servisna rutina** (interrupt service routine).
Dokler si operacijski sistem deli programski in strojne vire z uporabniki moramo poskrbeti, da morebitna napaka v uporabniškem programu vpliva samo na ta program. Napake kot so neskončna zanka v katero se je zapletel program moramo odpraviti drugače.

2.2.1. Dvojni režim delovanja

Za pravilno delovanje je potrebno zaščititi operacijski sistem, vse programe in pripadajoče podatke pred programom z napakami.
Najmanj kar potrebujemo sta dva ločena režima delovanja:

- **uporabniški način** (user mode)
- **sistemski način** (system mode ali monitor mode ali supervisior mode ali privileged mode ali kernel mode)

Način v katerem trenutno operiramo določa poseben **bit** (mode bit), ki je vgrajen v strojno opremo.
Vrednost bita 0 predstavlja sistemski način, 1 pa uporabniški način delovanja. Tako lahko ločimo opravilo, ki ga izvaja uporabnik ali operacijski sistem.
Ko sistem izvaja uporabniški program je sistem v uporabniškem režimu delovanja. Ko uporabniški program zahteva servis od operacijskega sistema (preko sistemskega klica) mora preklopiti režim delovanja v sistemski. V tem režimu izvede sistemski klic in nazaj spremeni režim delovanja.



Ko se operacijski sistem zaganja se zaganja v sistemskem načinu – ni mogoče, da bi tule posegal uporabnik. Šele na koncu nalaganja jedra naloži uporabniške procese v uporabniškem načinu delovanja. Ko pride do prekinitve ali pasti strojna oprema preklopi v sistemski režim delovanja s postavitvijo bita na 0. Predenj operacijski sistem preda kontrolo nazaj uporabniškemu procesu postavi bit na 1.
Dvojni način delovanja nam omogoča zaščito operacijskega sistema pred nezanesljivimi uporabniki. Preprečiti moramo možnost izvajanja **privilegiranih navodil** v uporabniškem načinu, ki bi povzročili škodo. V ta namen strojna oprema pusti izvajanje privilegiranih navodil samo v sistemskem načinu.
Ob poizkusu izvajanja privilegiranih navodil v uporabniškem načinu bi strojna oprema namesto izvedbe navodil signalizirala operacijski sistem preko pasti.
Uporabnik lahko izvede privilegirana navodila, vendar samo tiste, ki jih ima vgrajene operacijski sistem. Zahteva po izvedbi privilegiranih navodil se tako vrši samo preko sistemskega klica (system call).

Primer:
MS-DOS je bil napisan za procesor Intel 8088, ki nima "mode bita", torej tudi operacijski sistem ne pozna dvojnega načina dela.
Novejši procesorji Intel Pentium imajo vgrajen "mode bit" in novejši operacijski sistemi izkoriščajo ta bit za zaščito OS.

2.2.2. Časovnik

Zagotoviti moramo, da operacijski sistem drži vajeti v svojih rokah in kontrolira procesor. Preprečiti moramo uporabniškemu programu, da bi se ulovil v neskončno zanko, da nebi klical sistemske servise in nenazadnje da nebi nikoli predal kontrole nazaj operacijskemu sistemu.
Da zagotovimo tako zaščito uporabimo **časovnik** (timer). Časovnik naj bi prekinil delovanje po določenem času (npr. 60 krat v s). Časovnik upravlja operacijski sistem preko privilegiranih navodil. Ko preklopi v uporabniški režim operacijski sistem vedno nastavi časovnik, tako, da v primeru napake na programu, se sproži prekinitev preko časovnika in kontrola se prenese v operacijski sistem, ki obravnava prekinitev kot usodna napaka (fatal error) in program prekine.

2.3. Upravljanje procesov

Program ne dela nič dokler njegovih instrukcij ne izvaja procesor. Programu v izvajanju pravimi **proces**. Proces, ki deluje po principu večopravilnosti je na primer prevajalnik (compiler). Urejevalnik besedil, ki ga izvaja uporabnik na računalniku je proces. Sistemsko opravilo, kot je pošiljanje podatkov tiskalniku.
Proces potrebuje določene strojne vire, ki vključuje procesor, pomnilnik, datoteke in V/I naprave. Viri so dodeljeni procesu, ko se zažene ali ko jih med izvajanjem potrebuje.
Procesi med izvajanjem potrebujejo tako fizične kot tudi logične vire. Logični viri so inicializacijski in vhodni podatki, ki jih proces potrebuje ob zagonu in izvajanju.
Kot primer si vzemimo proces, ki ima nalogo prebrati status datoteke in ga izpisati na ekran terminala. Procesu najprej podamo vhodni podatek – ime datoteke, ta izvede potrebne instrukcije in sistemske klice, ki poiščejo datoteko, preberejo njen status in prikaže status na ekran. Ko proces opravi nalogo ga operacijski sistem pobriše in sprosti vire, ki jih je uporabljal.
Program sam po sebi je shranjen na disku in je **pasivna entiteta**, medtem ko je proces program, ki se izvaja in je **aktivna entiteta**. Vsak proces ima **programski števec**, ki kaže na naslednjo instrukcijo, ki se bo izvedla. Izvajanje procesa poteka tako, da procesor izvede eno instrukcijo za drugo po taktu programskega števca.
Proces je enota dela v sistemu. Sistem se sestoji iz **sistemskih** in **uporabniških** procesov. Programski procesi izvajajo uporabniško kodo, sistemski procesi pa sistemsko kodo. Procesi se lahko izvajajo istočasno, tako, da se usmerja delo procesorja enkrat na en proces drugič na drugi proces, itd..

Operacijski sistem je odgovoren za naslednje aktivnosti v zvezi z upravljanjem procesov:
- ustvarjanje in brisanje procesov
- ustavljanje in ponovno aktiviranje procesov
- zagotoviti mehanizme za sinhronizacijo med procesi
- zagotoviti mehanizme za komunikacijo med procesi
- zagotoviti mehanizme za odpravo smrtnih objemov

2.4. Upravljanje pomnilnika

Glavni pomnilnik je osrednji del delovanja modernega sistema. Glavni pomnilnik je veliko polje računalniških besed ali bytov (B) (1B = 8bitov!). Vsak beseda v pomnilniku ima svoj **naslov**. Glavni pomnilnik je hitro in direktno dostopno odložišče, kjer si procesor in V/I naprave izmenjujejo podatke in informacije. Po **Von Neumanovi** arhitekturi procesor prebere instrukcijo iz glavnega pomnilnika (cikel nalaganja instrukcije) in bere in piše podatke v glavni pomnilnik (cikel branja in pisanja podatkov). Glavni pomnilnik je edina večja naprava za shranjevanje podatkov, ki je **direktno** naslovljiva in dostopna procesorju.
Če želi procesor izvajati podatke, ki so na magnetnem disku, mora najprej prenesti v glavni pomnilnik (procesor izvede V/I klic) in jih preko glavnega pomnilnika uporabiti pri izvajanju.
Program, ki se želi izvajati mora biti razdeljen na absolutne naslove in naložen v glavnem pomnilniku. Ko se program izvaja proizvaja absolutne naslove, da ve procesor kam iskati instrukcije in podatke med izvajanjem programa. Ko se program zaključi se prostor v glavnem pomnilniku, ki ga je zasedala programska koda sprosti. Naslednji program se lahko naloži.
Da se izboljša izraba procesorja in hitrost odziva sistema uporabniku mora sistema držati več programov v glavnem pomnilniku. Pojavi se potreba po upravljanju pomnilnika.
Operacijski sistem je odgovoren za naslednje aktivnosti pri upravljanju pomnilnika:

- sledenje kateri deli pomnilnika so prosti in kdo jih zaseda
- odločanje katere procese prenesti v glavni pomnilnik in katere iz glavnega pomnilnika
- dodeljevati in sprostiti prostor v glavnem pomnilniku, ko je potrebno

2.5. Upravljanje shranjevanja

Operacijski sistem poskrbi, da uporabniki enolično in logično pridejo do enot za shranjevanje informacij. Zato operacijski sistem definira enoto za shranjevanje informacij – datoteko. Operacijski sistem dodeli datoteki fizikalne dele medija za shranjevanje.

2.5.1. Upravljanje z datotečnim sistemom

Datotečni sistem je najbolj vidna komponenta operacijskega sistema. Računalniki lahko shranijo informacije na več vrst fizičnih medijev (magnetni disk, optični disk, magnetni trak,..). Vsaka naprava ima svoje lastnosti in značilnosti, katere kontrolira pogon (pogon magnetnega diska, CD pogon, pogon magnetnega traku, ...). Lastnosti naprave zajemajo dostopni čas, prenosne hitrosti in metoda dostopa (zaporedni in naključni dostop).
Datoteka je zbirka povezanih informacij, ki jo je ustvaril lastnik datoteke. Datoteka lahko predstavlja **program** (v izvorni ali objektni obliki) ali **podatke**. Podatki v datoteki so lahko v numerični, abecedni, alfanumerični ali binarni obliki.
Datoteke so organizirane v **direktorijih**. V primeru več uporabnikov, se dostop do datoteke kontrolira (lastništvo in način dostopa do datoteke).
Operacijski sistem je odgovoren za naslednje aktivnosti datotečnega sistema:

- ustvarjanje in brisanje datotek in direktorijev
- podpora enostavni uporabi datotek in map
- dodeljevanje prostora datotekam na sekundarnem mediju (magnetnem disku)

- varnostno kopiranje datotek na zanesljiv in trajen shranjevalni medij

2.5.2. Upravljanje z masovnimi napravami za shranjevanje

Ker je glavni pomnilnik premajhen, da bi shranil vse programe in podatke in ker glavni pomnilnik izgubi vsebino, ko mu zmanjka energije mora sistem zagotoviti **sekundarni** medij za zanesljivo in permanentno shranjevanje programov in podatkov. Moderni računalniški sistemi uporabljajo **magnetni disk** za shranjevanje med delovanjem sistema (on-line shranjevanje). Večina programov (prevajalniki, zbirni jeziki, urejevalniki besedil, urejevalniki in oblikovalniki) so trajno shranjeni na magnetnem disku, dokler jih sistem ne prenese v glavni pomnilnik. Magnetni disk se nato uporabi kot izvor in ponor podatkov pri procesiranju. Zato je upravljanje naprave za shranjevanje med delovanjem zelo pomembno. Operacijski sistem je odgovoren za naslednje aktivnosti upravljanja s shranjevalnimi napravami:

- upravljanje prostega prostora na sekundarnem mediju
- dodeljevanje prostega prostora
- razvrščanje na disku

Obstajajo tudi **terciarne** naprave za varnostno shranjevanje podatkov, malo rabljenih podatkov in arhivov podatkov. Običajno so te naprave počasnejše (optični disk, magnetni trak). Terciarni mediji so lahko tipa WORM (enkrat zapiši, večkrat beri - Write Once, Read Many) in RW (beri, piši – Read Write). Terciarni mediji ne vplivajo na hitrost sistema in je v večini primerov puščeno upravljanje aplikacijskim programom. Operacijski sistem tula bolj skrbi za priklop medija v datotečni sistem, ...

2.5.3. Upravljanje hitrega pomnjenja (caching)

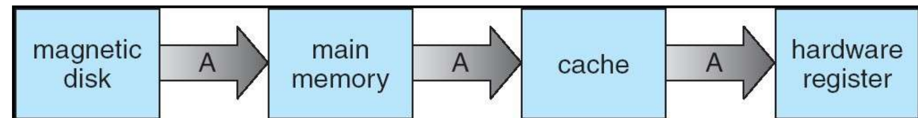
Hitro pomnjenje je pomemben princip računalniških sistemov. Informacija je običajno shranjena na nekem mediju za shranjevanje (npr. glavni pomnilnik). Ko potrebujemo določeno informacijo gremo najprej pogledat v hitri pomnilnik naprave. Če je v hitrem pomnilniku jo uporabimo, če je ni jo skopiramo v hitri pomnilnik s predpostavko, da jo bomo po možnosti ponovno potrebovali (tehnika se imenuje pametno hitro pomnjenje – **smart caching**).

V praksi so to interni registri naprav tipa SRAM (statični pomnilnik – ne potrebuje osveževanja in ima zato dostopni čas 10 ns (npr. indeksni register)) in predstavljajo hitri pomnilnik za glavni pomnilnik, ki je tipa DRAM (dinamični pomnilnik – potrebuje osveževanje (dostopni čas 60 ns). Brez hitrega pomnilnika bi procesor porabil več urnih ciklov, da bi podatek prenesel iz glavnega pomnilnika v registre procesorja. Iz enakih razlogov imajo tudi druge naprave v shranjevalni verigi hitre pomnilnike.

Ker so hitri pomnilniki majhnih velikosti in dragi je upravljanje hitrega pomnilnika pomemben in zahteven del operacijskega sistema.

Level	1	2	3	4
Name	registers	cache	main memory	disk storage
Typical size	< 1 KB	> 16 MB	> 16 GB	> 100 GB
Implementation technology	custom memory with multiple ports, CMOS	on-chip or off-chip CMOS SRAM	CMOS DRAM	magnetic disk
Access time (ns)	0.25 – 0.5	0.5 – 25	80 – 250	5,000.000
Bandwidth (MB/sec)	20,000 – 100,000	5000 – 10,000	1000 – 5000	20 – 150
Managed by	compiler	hardware	operating system	operating system
Backed by	cache	main memory	disk	CD or tape

Prenos informacije med nivoji shranjevanja se lahko izvaja strojno ali preko operacijskega sistema. Na primer prenos podatkov iz hitrega pomnilnika v procesor poteka pod okriljem strojne opreme, brez posredovanja operacijskega sistema, medtem ko je prenos podatka iz magnetnega diska v glavni pomnilnik funkcija operacijskega sistema.
V hierarhični shranjevalni organizaciji se lahko en podatek pojavi na več nivojih shranjevalne organizacije. Primer celoštevilčni podatek A, ki ga moramo povečati za 1 se nahaja v datoteki B in datoteka B se nahaja na magnetnem disku. Operacija inkrementiranja poteka tako, da se izvede V/I operacija, ki kopira blok na disku, kjer je podatek A v glavni pomnilnik. Sledi operacija kopiranja podatka A v hitri pomnilnik in nato v interne registre. V tem primeru podatek A domuje na več mestih (magnetni disk, glavni pomnilnik, hitri pomnilnik in interni strojni registri). Ko se v procesorju, natančneje v



registrih izvede operacija povečanja vrednosti za ena mora podatek prepotovati enako pot nazaj in se zapisati v blok na magnetnem disku.
V sistemih, kjer se naenkrat izvaja en proces z vrednostjo podatka ni problemov. Nasprotno v večopravilnih sistemih, kjer procesor preklaplja med procesi je potrebno paziti, da nebi nek drug proces prebral neshranjen podatek na magnetnem disku (proces, ki inkrementira A, se prekine, ko še ni shranil nove vrednosti na magnetni disk in naslednji proces prebere z diska "star", nepravilen podatek).

2.6. Varnost in zaščita

Če ima sistem več uporabnikov in se procesi izvajajo istočasno mora operacijski sistem kontrolirati dostop do podatkov. Mehanizmi torej omogočajo, da so datoteke, segmenti v pomnilniku, procesor, registri strojne opreme dostopni samo procesom, ki imajo dovoljenje operacijskega sistema.

- strojna oprema za dodeljevanje prostora v pomnilniku dovoljuje procesu da uporablja samo svoj naslovni prostor
- časovnik onemogoča, da bi proces prevzel popoln nadzor nad procesorjem
- do kontrolnih registrov strojnih naprav ne more dostopati uporabnik preko uporabniških procesov ampak samo operacijski sistem

Zaščita je torej vsak mehanizem, ki kontrolira dostop procesov ali uporabnikov do virov, ki jih definira operacijski sistem.
Sistem ima lahko zaščito, vendar lahko še vedno pride do zlorabe. Na primer uporabnik lahko ukrade identifikacijske podatke druge osebe in s tem povzroči škodo (kopiranje, brisanje podatkov brez vednosti lastnika). Naloga **varnosti** je preprečiti zunanje in

notranje napade na sistem. Napadi se širijo in obsegajo viruse, črve, DoS napade, kraja računalniške identitete, ... Varnost lahko zagotavlja operacijski sistem ali dodaten uporabniški program (AntiVirus, SpyWare, Požarni zid...).
Varnost in zaščita sistema mora omogočati razlikovanje med uporabniki sistema. Operacijski sistemi tako hranijo spisek uporabniških imen in asociiranih uporabniških identifikacij (User ID v Unixu in Security ID v MS Windows NT). UID je eden za vsakega uporabnika.

2.7. Porazdeljeni sistemi

Porazdeljen sistem je zbirka fizično ločenih sistemov, ki so med seboj povezani preko omrežja in omogočajo uporabnikom dostop do različnih virov, ki jih sistem vzdržuje. Porazdeljeni sistemi so zmožni komunicirati in s tem deliti računska opravila. **Omrežje** je komunikacijska pot med dvema ali več sistemi. Omrežje je odvisno od protokola (npr. TCP/IP), ki ga uporablja, razdalje med sistemi in prenosnimi mediji, ki sisteme povezuje. Omrežja se delijo glede na razdaljo med sistemi.

- LAN (Local Area Network) omrežje je značilno za porazdeljene sisteme, ki se nahajajo znotraj ene sobe, nadstropja ali stavbe.
- WAN (Wide Area Network) omrežje je značilno za porazdeljene sisteme, ki se nahajajo med stavbami, mesti ali celo državami. Kot primer vzemimo podjetje, ki ima širom sveta svoje poslovne enota in mora enote primerno povezati.
- MAN (Metropolitan Area Network) omrežje je značilno za porazdeljene sisteme, ki povezujejo zgradbe znotraj mesta.
- SAN (Small Area Network) – BlueTooth in mrežne naprave(802.11) uporabljajo brezžično komunikacijo na kratke razdalje in v osnovi ustvarijo majhno omrežje.

Medij je lahko bakrena žica, optična vlakna, brezžično komuniciranje (sateliti, mikrovalovi, komuniciranje preko radijskih valov.
Mrežni operacijski sistem je operacijski sistem, ki omogoča storitve kot so skupna raba datotek preko omrežja, komunikacijske sheme, ki omogočajo različnim procesom na različnih sistemih, da si izmenjujejo sporočila. Sistem, ki poganja mrežni operacijski sistem deluje avtonomno, vendar se zaveda omrežja in je zmožen komunicirati z drugimi mrežnimi računalniki. Porazdeljen operacijski sistem zagotavlja manj avtonomno okolje. Različni operacijski sistemi komunicirajo tako blizu, da se zdi, da samo en operacijski sistem kontrolira celotno dogajanje v omrežju.

2.8. Sistemi s posebnim namenom

2.8.1. Realno časovni vgrajeni sistemi

To je oblika namenskega operacijskega sistema, kjer se zaradi strogih časovnih omejitev zahteva usklajena obdelava nalog ali pretoka podatkov. To so sistemi, ki kontrolirajo znanstvene poizkuse, medicinsko obsevalni sistemi, industrijski kontrolni sistemi, bolj konkretno recimo sistem za elektronsko vbrizgavanje goriva.

2.8.2. Ročni sistemi

Sem spadajo osebni digitalni asistenti, kot so dlančniki, mobilni telefoni, ki se lahko

povežejo v omrežje, na primer Internet. Načrtovalci ročnih sistemov se spopadajo z mnogimi pastmi, ki jih prinašajo 12 cm x 7 cm velike napravice. Taki sistemi imajo malo spomina, majhno procesorsko moč, majhne tipke in zaslon. Operacijski sitem mora v takem primeru pomnilnik zelo efektivno uporabljati.

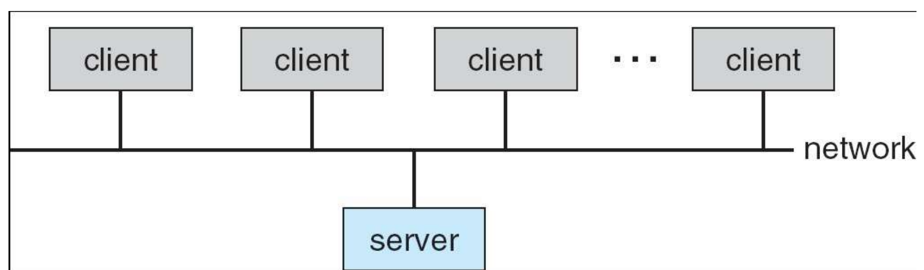
2.9. Računalniška okolja

2.9.1. Tradicionalno računalniško okolje

Se počasi izgublja. Zamislimo si tradicionalno pisarno izpred nekaj let, kjer je bil računalnik povezan v omrežje in izkoriščal mrežne datoteke in mrežno tiskanje. Dostop do oddaljenih računalnikov je bil redek, razdalje so premagovali z prenosnimi računalniki. Terminali, ki so bili povezani v velik računalnik (mainframe) so prevladovali v večjih podjetji, kjer je bilo še manj dostopov do oddaljenih računalnikov.
Današnji trend je usmerjen proti vse zagotavljanju vse več poti za dostop do računalniških okolij. **Spletne tehnologije** stiskajo meje tradicionalnemu računalniškemu okolju. Podjetja in ustanove vzpostavljajo **portale**, ki omogoča dostop do internih strežnikov podjetja in ustanove. Mrežni računalniki so v osnovi terminali, ki razumejo spletno računalniško komuniciranje.
Doma je imela večina ljudi počasno internetno povezavo do ponudnika internetnih storitev (ISP), kjer se je lahko povezal v službo, internet, ... Danes so hitrosti narasle (ADSL, Kabelski dostop, ...) in dajejo uporabnikom hitrejši dostop do več podatkov. Hitre podatkovne povezave omogočajo, da si ljudje postavijo spletne strani.

2.9.2. Okolje odjemalec-strežnik

Ker so osebni računalniki postali cenejši, močnejši in hitrejši se oblikovalci sistemov oddaljujejo arhitekture centralnega sistema povezanega s terminali. Terminali povezani v centralen velik računalnik se nadomeščajo z osebnimi računalniki. Vloga in servisi centralnega računalnika se nadomešča z osebnimi računalniki. Kot rezultat osebnega računalnika, ki nadomešča centralni računalnik je računalnik imenovan **strežnik**. Strežnik streže zahteve, ki jih ustvarja **odjemalec**.



Strežniški sistem lahko razdelimo na **računski strežnik** in **datotečni strežnik**.

- računski strežnik omogoča vmesnik, preko katerega odjemalec pošlje zahtevo za izvajanje nekega programa na strežniku (npr. branje podatka iz baze podatkov). Strežnik zahtevi ugodi, tako da izvede program, ki prebere podatek in ga posreduje nazaj odjemalcu
- datotečni strežnik omogoča datotečni vmesnik, kjer odjemalec lahko manipulira z datotekami na strežniku (npr. spletni strežnik, kjer odjemalec pošlje zahtevo za branje

html dokumenta na spletnem strežniku, strežnik html dokument poišče in ga posreduje odjemlcu)

2.9.3. Okolje enakovrednih sistemov

To je še ena oblika porazdeljenih sistemov imenovana tudi peer-to-peer ali P2P omrežje. V tem modelu se odjemalci in strežniki ne razlikujejo med seboj. Vsak član P2P omrežja je lahko v vlogi odjemalca, strežnika ali oboje hkrati, odvisno od tega ali zahteva storitev ali streže zahtevi po storitvi. P2P model prinaša prednosti pred modelom strežnik odjemalec, saj lahko v P2P modelu opravljamo servis na več strežnikih hkrati, medtem, ko v modelu odjemalec-strežnik vsi odjemalci dostopajo do enega strežnika (lahko postane ozko grlo).
Za delovanje v P2P sistemu se mora sistem prijaviti v omrežje P2P. Ko se v omrežje prijavi in postane enakopraven član P2P omrežja lahko začne pošiljati zahteve drugim sistemom in servirati druge sisteme.
Servise lahko koristimo na dva različna načina:

- ko se sistem pridruži P2P omrežju, se njegov servis prijavi v centralnem poizvedovalnem servisu na omrežju. Vsak član omrežja, ki zahteva servis najprej vpraša centralni sistem na katerem sistemu se nahaja željen servis in nato se sistem poveže z drugim-i sistemi.
- sistem, ki deluje kot odjemalec najprej pošlje poizvedbo (broadcast) po vseh sistemih v P2P omrežju kdo omogoča željen servis. Ko dobi od sistemov odgovor se s temi sistemi direktno poveže in koristi servis, ki ga sistemi ponujajo. Tak postopek zahteva protokol imenovan **discovery protocol**.

Predstavnik prvega načine je Napster, drugega pa Gnutella. Servisa omogočata izmenjavo datotek. Prvi način omogoča večjo kontrolo nad prenašanjem datotek, saj se tule postavlja vprašanje legalnosti in avtorskih pravic prenesenih datotek.

2.10. Vprašanja za ponavljanje

1. Opiši 4 skupine v katere bi razdeli računalniški sistem!

2. Na kakšen način lahko pride do prekinitve delovanja procesorja.

3. Za katere tipe operacij je uporaben direkten dostop do pomnilnika (DMA)? Obrazloži odgovor!

4. Kakšna računalniška arhitektura omogoča hiter prenos podatkov med glavnim pomnilnikom in lokalnim pomnilnikom kontrolerja V/I naprave?

5. Do katerih pomnilnih naprav lahko procesor dostopa direktno?

6. Zakaj ni mogoče primarnega ali glavnega pomnilnika izkoristiti kot edinega predstavnika za shranjevanje podatkov?

7. Kaj so primarni, kaj sekundarni in kaj terciarni viri za shranjevanje podatkov?

8. Opiši hierarhijo naprav za shranjevanje podatkov. Kaj je značilno za vsako stopnjo v hierarhiji?

9. Kaj je značilno za enoprocesorski in kaj za večprocesorski sistem?

10. Kakšne so prednosti večprocesorskega sistema in kakšna dva modela večprocesorskega sistema poznaš?

11. Katere naloge mora opravljati večopravilen operacijski sistem?

12. Kaj je past(trap) in kdo jo povzroči?

13. Na kakšne načine in kako lahko zaščitimo nemoteno delovanje računalniškega sistema?

14. Kaj je proces?

15. Za katere aktivnosti je OS odgovoren pri upravljanju procesov?

16. Za katere aktivnosti je OS odgovoren pri upravljanju pomnilnika?

17. Za katere aktivnosti je OS odgovoren pri upravljanju shranjevanja?

18. Kdaj je smiselno uporabljati hitri pomnilnik (cache)? Zakaj nebi namesto počasnega trdega diska uporabili kar hitri pomnilnik (cache)?

19. Kaj je varnost in kaj zaščita sistema?
20. Kakšne vrste zaščite računalniškega sistema poznamo?
21. V kakšni obliki lahko varujemo sistem pred zunanjimi in notranjimi zlorabami?

3. Strukture operacijskih sistemov

Operacijski sistem zagotavlja okolje v katerem se lahko izvajajo programi. Operacijski sistem lahko pogledamo s strani

- uporabnika, ki uporablja operacijski sistem in programe, ki se izvajajo v njegovem okolju,
- programerja, ki ustvarja, prevaja in testira programe za uporabnike in
- razvijalci operacijskih sistemov, kjer morajo biti cilji operacijskega sistema določeni predenj se začne razvoj.

3.1. Sistemske komponente

Kompleksen operacijski sistem lahko naredimo, le če ga razdelimo na logične dele. Veliko operacijskih sistemov se nagiba k podpori naslednjih delov operacijskega sistema:

- Upravljanje procesov
- Upravljanje glavnega pomnilnika
- Upravljanje datotečnega sistema
- Upravljanje sistema vhodno izhodnih naprav
- Upravljanje sekundarnega sistema za shranjevanje (magnetni disk)
- Omrežje
- Sistem zaščite
- Sistem ukaznega tolmača

3.1.1. Upravljanje procesov

Program se izvaja, če njegovo kodo izvaja procesor. Programu v izvajanju pravimo **proces**. Proces je lahko urejevalnik besedil, ki ga je pognal uporabnik, prevajalnik programske kode, sistemsko opravilo, ki pošlje tiskalniku opravilo, ki ga mora natisniti.
Proces potrebuje za izvajanje nalog **sredstva**, kot so procesor, pomnilnik, datoteke in V/I naprave.
Vsak proces ima svoj **programski števec**, ki kaže na naslednjo instrukcijo, ki se bo izvedla. Izvajanje procesa poteka zaporedno – procesor izvaja eno instrukcijo za drugo dokler izvajanje procesa ne prode do konca. Takrat se tudi sprostijo vsa prej zasedena sredstva. Proces lahko ima niti, kar pomeni, da ima vsaka nit svoj programski števec. (npr. urejevalnik besedil spremlja in prikazuje vnose uporabnika in hkrati tudi preverja črkovanje. To bi bil primer procesa, ki vsebuje dve niti).
Proces je tudi enota dela v sistemu. Sistem je sestavljen iz več procesov, ki se ločijo na sistemske procese (izvajajo kodo operacijskega sistema) in uporabniške procese (izvajajo kodo uporabniških programov).
Vsi procesi se lahko izvajajo istočasno, tako, da se izvaja en proces nato drugi, itd.. To preklapljanje procesorja med procesi je zelo hitro, tako da uporabnik, ki poganja en proces misli, da je procesor v celoti posvečen njegovemu procesu.
Operacijski sistem je odgovoren za naslednje aktivnosti v upravljanjem procesov:

- ustvarjanje in brisanje sistemskih in uporabniških procesov
- začasno ustaviti in ponovno pognati proces
- zagotoviti mehanizem sinhronizacije med procesi
- zagotoviti mehanizem komunikacije med procesi

- zagotoviti mehanizem za rokovanje s smrtnimi objemi

3.1.2. Upravljanje glavnega pomnilnika

Osnovni pomnilnik je veliko polje bytov ali računalniških besed. Vsaka beseda ima svoj naslov. Osnovni pomnilnik je skladišče podatkov, ki je hitro in direktno dosegljivo procesorju in V/I napravam. Procesor prebere instrukcijo iz glavnega pomnilnika pri **ciklu prenosa instrukcije** iz pomnilnika v procesor. Procesor piše in bere v/iz glavnega pomnilnika, ko izvaja **cikel prenosa podatkov** iz/v pomnilnik (v Von Neuman-ovi arhitekturi). V/I operacije, ki so izvedene preko **direktne povezave z glavnim pomnilnikom** (**DMA**), tudi berejo in pišejo v glavni pomnilnik. Glavni pomnilnik je v splošnem edina večja naprava za shranjevanje podatkov, ki je **direktno dostopna** procesorju. Program, ki naj se izvede, se mora najprej prenesti iz magnetnega diska v glavni pomnilnik in potem šele začne izvajati.
Vrstice programa dobijo absolutne naslove v glavnem pomnilniku, kjer je program naložen. Ko se program zaključi se njegov prostor v pomnilniku določi kot prost za nadaljnjo uporabo.
Da izboljšamo izrabo procesorja in hitrost sistema moramo hraniti več programov v glavnem pomnilniku, to pa zahteva upravljanje z prostorom pomnilnika.
Operacijski sistem je v tem primeru dolžan skrbeti za:

- vodenje evidence o zasedenem prostoru v pomnilniku in kdo prostor zaseda
- odločanje, kateri proces naj se naloži, ko se pomnilniški prostor sprosti
- zasedati in sprostiti pomnilniški prostor ko je to potrebno

3.1.3. Upravljanje datotečnega sistema

Datotečni sistem je gledano s strani uporabnika najbolj viden del operacijskega sistema. Računalniški sistem lahko shranjuje informacije na razne medije za trajno shranjevanje (magnetni disk, optični disk, magnetni trak, električni disk, ...). Vsak od naštetih medijev ima različne karakteristike in načine organizacije. Lastnosti medijev vključujejo hitrost dostopa, hitrost prenosa, vrsta dostopa (direktni, zaporedni), kapaciteta hranjenja podatkov, itd....
Da se abstraktnost sistema za shranjevanje prekrije, operacijski sistem določi človeku prijazen pogled na shranjevalni medij in določi logično enoto shranjevanja – **datoteko**. Datoteka je zbirka povezanih informacij, ki jih ustvari lastnik datoteke. V splošnem lahko datoteka predstavlja program, prevajalnik ali podatke. Datoteke so organizirane v direktorije, direktoriji pa v razdelke in razdelki v fizične enote za shranjevanje podatkov. Datoteke, ki so dostopne več uporabnikom morajo biti ustrezno zaščitene.
Operacijski sistem je odgovoren za:

- ustvarjanje in brisanje datotek in direktorijev
- dodeljevanje datotek na medij za shranjevanje
- prenos datotek na stabilen medij – virtualni pomnilnik

3.1.4. Upravljanje V/I sistema

Namen operacijskega sistema je, da skrije uporabniku posebnosti, specifiko in raznolikost V/I naprav (Unix skrije posebnosti V/I naprav z V/I podsistemom, ki vsebuje upravljanje z raznimi vrstami pomnilnikov, vmesnik za gonilnike in gonilnike za naprave). Samo gonilnik

pozna posebnosti naprave.

3.1.5. Upravljanje s sekundarnim pomnilnikom

Glavni namen sistema je, da poganja programe. Programi so med izvajanjem v glavnem pomnilniku ali **primarnem pomnilniku**. Ker je glavni pomnilnik premajhen za hranit vse podatke in programe in se vsebina primarnega pomnilnika izgubi, če zmanjka energije je potrebno zagotoviti veliko kapaciteto in trajno shranjevanje podatkov. To funkcijo prevzema **sekundarna naprava** za shranjevanje podatkov in je danes običajno magnetni disk. Veliko programov (zbirniki, prevajalniki, urejevalniki, urejevalniki besedil, ...) so najprej shranjeni na magnetnem disku in ko se naložijo v glavni pomnilnik uporabljajo magnetni diska kot izvor ali ponor za njihovo procesiranje (datoteka/odpri, datoteka/shrani, datoteka/shrani kot, ...).
Operacijski sistem skrbi za :

- upravljanjem prostega prostora,
- dodeljevanje prostora enotam zapisa – blokom
- razvrščanje na disku

3.1.6. Omrežje

Porazdeljen sistem je zbirka procesorjev, ki si ne delijo pomnilnika, perifernih naprav ali takta. Vsak procesor ima svoj pomnilnik in takt, ter procesorji komunicirajo med seboj preko različnih komunikacijskih linij (hitra vodila ali omrežja).
Procesorji so povezani preko **komunikacijskega omrežja**. Omrežje je lahko popolnoma ali delno povezano. Porazdeljen sistem združuje fizično ločene sisteme v ne koherentni sistem, ki omogoča uporabniku, da dostopa do sredstev, ki jih sistem vsebuje.
Dostop do skupne rabe poveča hitrost izvajanja določene naloge, funkcionalnost, dostopnost podatkov, povečana zanesljivost.
Sistemi uporabljajo skupni jezik – **protokol**, da si izmenjujejo podatke. Kot izboljšava in prilagoditev protokola za izmenjavo datotek (file transfer protocol - FTP) in protokola za mrežni datotečni sistem (network file system - NFS) je nastal protokol **http**. Protokol omogoča WWW storitev, ki omogoča dostopnost informacij brez avtentikacije in se uporablja za komunikacijo med spletnim strežnikom in spletnim brskalnikom. Spletni brskalnik pošlje samo zahtevo za informacijo, spletni strežnik pa jo posreduje v obliki teksta, slik, povezave, ...

3.1.7. Sistem zaščite

Če ima sistem več uporabnikov in dovoli, da se istočasno izvaja več procesov, morajo biti procesi zaščiteni eden od drugega. V ta namen mehanizmi omogočajo, da se na datotekah, segmentih pomnilnika, procesorju in ostalih sredstvih delajo le procesi, katerim da dovoljenje operacijski sistem. Na primer strojna oprema za naslavljanje pomnilnika zagotovi, da se proces izvaja samo znotraj pomnilniškega segmenta, ki mu je bil dodeljen. Registri kontrolerjev V/I naprav niso dosegljivi uporabniku, zato je integriteta V/I naprav zaščitena.
Zaščita je torej vsak mehanizem, ki kontrolira dostop programov, procesov ali uporabnikov do sredstev, ki jih je definiral računalniški sistem.

3.1.8. Sistem ukaznega tolmača

Eden najpomembnejših sistemskih programov operacijskega sistema je **tolmač ukazov**, ki jih uporabnik posreduje sistemu. Tolmač ukazov je vmesnik med uporabnikom in operacijskim sistemom. Nekateri operacijski sistemi vgrajujejo tolmača v jedro operacijskega sistema, drugi kot MS-DOS in UNIX, pa uvrščajo tolmača v program, ki se inicializira, ko se neko opravilo zažene (pri paketnih sistemih) ali ko se uporabnik prijavi v sistem (pri večopravilnih sistemih). Ukazi se operacijskemu sistemu lahko podajajo preko kontrolnih stavkov (*copy *.* a:*, *mkdir test*) v **ukazni lupini** ali preko grafičnega vmesnika (uporaba miške in grafičnih elementov)
V primeru podajanja ukazov preko grafičnega vmesnika je potrebna uporaba miške in izbire grafičnih elementov. Z različnimi kombinacijami tipk in klikov na različne grafične elemente podajamo operacijskemu sistemu ukaze. Način je prijazen do uporabnika, a ne vedno učinkovit.
V primeru podajanja ukazov preko ukazne lupine (shell-a) se ukazi vpisujejo ročno, se ukazi in rezultati prikazujejo na zaslonu. Tak način podajanja ukazov je težji za se naučit, zato pa bolj obširen in zmogljiv.

3.2. Servisi operacijskega sistema

Operacijski sistem zagotavlja okolje v katerem se izvajajo programi. Zagotovi servise programom in uporabnikom teh programov.
Servisi pomagajo programerju pri programiranju in uporabnikom pri uporabi sistema:

- **izvajanje programa** – sistem mora biti sposoben naložiti program v pomnilnik in ga začeti izvajati. Program mora biti sposoben končati izvajanje normalno ali nenormalno (če je prišlo do napake).
- **V/I operacije** – tekoči program lahko zahteva V/I operacijo, ki zajame datoteko na disku ali V/I napravo. Zaradi zmogljivosti in zaščite, uporabnik ne more direktno dostopati do V/I naprave, zato mora uporabnik to nalogo dodeliti operacijskemu sistemu.
- **rokovanje z datotečnim sistemom** – programi lahko izvajajo vse operacije nad datotekami in mapami (berejo, pišejo, preimenujejo, brišejo, ...).
- **komunikacije** – v veliko primeri pride do situacije, ko morata dva procesa med seboj komunicirati – si izmenjevati informacije. Komunikacija lahko poteka lokalno preko glavnega pomnilnika ali preko računalniškega omrežja. V prvem primeru govorimo o **skupni rabi pomnilnika**, v drugem pa o **pošiljanju sporočil**.
- **detekcija napak** – operacijski sistem se mora zavedati možnih napak. Napake se lahko zgodijo v procesorju ali pomnilniku (kot napaka pomnilnika ali prekinitev napajanja), v V/I napravah (kot paritetna napaka na trakovih, prekinitev mrežne povezave med komunikacijo) ali zmanjkanje papirja v tiskalniku. Zgoraj smo našteli napake na strojni opremi, napake se lahko zgodijo tudi na programski opremi kot na primer aritmetično prekoračenje, deljenje z 0, poizkus naslavljanja zaščitenega naslova v pomnilniku, zasedanje preveč procesorskega časa.
  Za vsak tip napake mora operacijski sistem izvršiti določeno akcijo, da zagotovi kljub napaki nemoteno delovanje celotnega sistema.

Obstajajo tudi servisi, ki pomagajo k učinkovitosti samega sistema.

- **dodeljevanje sredstev** – ko je v sistemu več uporabnikov ali več procesov, ki se

izvaja istočasno mora operacijski sistem zagotoviti pravično in učinkovito dodeljevanje in izrabo sredstev kot so procesor (razvrščanje procesov, ki čaka na procesor), glavni pomnilnik (upravljanje z pomnilnikom kot je dodeljevanje prostega prostora, itd...),

- **opazovanje sistema** – beležiti želimo kateri uporabniki uporabljajo sredstva, koliko in kako. Slika uporabe prispeva k boljši postavitvi sistema za določen tip uporabnikov oziroma, da se uporabnikom zaračuna za uporabo sistema.
- **zaščita in varnost** – Lastniki sistema želijo, da imajo delovanje sistema in uporabo sredstev na sistemu pod kontrolo. Ko se ločeni procesi izvajajo istočasno ne sme priti do tega da bi en proces posegal v drugega oziroma v delovanje operacijskega sistema.
  Varnost sistema zahteva, da se vsak uporabnik sistema identificira (ponavadi z uporabniškim imenom in geslom) in tako dobi kontroliran dostop do določenih sredstev sistema.
  Veriga je tako močna kot njen najšibkejši člen!

3.3. Uporabniški vmesnik do operacijskega sistema

Obstajata dva načina, preko katerega uporabnik komunicira z operacijskim sistemom.
En način uporablja čisti tekstovni pristop – **CLI** (Command Line Interface) ali tolmač ukazov, ki omogoča, da uporabnik direktno vpisuje ukaze, katerim servira operacijski sistem.
Drugi način omogoča uporabniku, da komunicira z operacijskim sistemom preko grafičnega vmesnika – **GUI** (Graphical User Interface).

3.3.1. Ukazni tolmač (CLI)

Nekateri operacijski sistemi vsebujejo tolmača ukazov v jedru, drugi pa obravnavajo ukaznega tolmača kot poseben program, ki teče kadar se neko opravilo zažene ali ko se uporabnik prijavi v sistem (Windows XP, Linux).
Na sistemih z več ukaznimi tolmači se tolmači imenujejo lupine (shells). Linux sistemi vsebujejo več lupin (Bourne-Again Shell ali BASH lupina, C shell ali CSH lupina)

Najpomembnejši program operacijski sistem je **ukazni tolmač** (command interpreter). Naloga tolmača je, da sprejme in izvede ukaz, ki je bil podan od uporabnika preko tipkovnice. Poznamo dva načina izvedbe ukaznega tolmača.
Prva izvedba je ta, da tolmač vsebuje kodo, ki pripada določenemu ukazu (command.com ali cmd.com v Windowsih) (ukaz *delete* bo v ukaznemu tolmaču skočil na sekcijo, kjer nastavi parametre in izvede sistemski klic za brisanje datoteke). V tem primeru je število ukazov, ki jih tolmač pozna odvisen od velikosti ukaznega tolmača.
Druga izvedba, ki jo uporablja Unix/Linux je ta, da implementira veliko ukazov preko sistemskih programov. V tem primeru tolmač ne pozna nobenega ukaza ampak podan ukaz tolmaču pomeni, da poišče datoteko (sistemski program) s tem imenom, jo prenese v pomnilnik in jo izvede.
Ukaz za brisanje datoteke v Unixu/Linuxu:

rm G

bo poiskal datoteko z imenom *rm*, jo naložil v pomnilnik in jo z parametrom G izvedel. Kaj bo naredil ukaz rm je izključno odvisno od kode, ki je zapisana v datoteki *rm*. V tem primeru programerji lahko dodajajo, popravljajo in spreminjajo datoteko *rm*, brez, da bi

vplivali na sistem.

3.3.2. Grafični uporabniški vmesnik (GLI)

Drugi način, kako uporabniku omogočiti interakcijo z operacijskim sistemom je preko uporabniku prijaznega grafičnega vmesnika (Graphic User Interface – GUI). Grafični vmesnik uporablja okenske elemente (okna, ikone, menije, ...) in miško za interakcijo med uporabnikom in operacijskim sistemom. Grafični vmesnik vsebuje metaforično namizje, kjer miškin kazalec in njegova pozicija kaže na slike ali ikone, ki predstavljajo programe, datoteke, mape, bližnjice in sistemske funkcije. Odvisno od pozicije miškinega kazalca se ob kliku izvede program, izbere objekt, ..., ali se preko menija izbere ukaz.

Lahko bi rekli, da Windows operacijski sistemi (WinXXxx,) vsebujejo v osnovi in primarno grafični vmesnik za interakcijo uporabnika z operacijskim sistemom. Ukazni tolmač ali ukazna lupina (command.com ali cmd.com) je le eden od mnogih programov, ki jih vsebuje poleg jedra operacijski sistem.
Unix/Linux, pa v osnovi in primarno uporablja ukazni tolmač za interakcijo uporabnika z operacijskim sistemom. Razvoj grafičnih odprtokodnih vmesnikov (K Desktop, GNOME Desktop) je omogočilo, da je tudi Linux operacijski sistem postal bolj dostopen povprečnim uporabnikom.

Izbira vmesnika je odločitev vsakega posameznika. V osnovi uporabniki Unix/Linux sistemov ima raje CLI, ker jim predstavlja zmogljiv in učinkovit vmesnik, medtem ko uporabniki Windows operacijskega sistema uporabljajo GUI vmesnik in nikoli ne zaženejo cmd.com ali command.com CLI vmesnika.
Na Linux platformi obstaja tudi odprtokodni **spletni grafični vmesnik** za nadzor in upravljanje operacijskega sistem – Webmin (www.webmin.com).

3.4. Sistemski klici

Sistemski klici predstavljajo vmesnik med procesom in operacijskim sistemom. Klici so ponavadi podani v zbirnem jeziku (asemblerju). Klici so lahko tudi v obliki višjih programskih jezikov in se klici generirajo na podlagi prej narejenih funkcij ali podrutin v obliki zbirnika.
Veliko jezikov, kot so C in C++ so zamenjali sistemsko programiranje z zbirnim jezikom. S temi jeziki lahko izvajamo sistemske klice direktno. Primer Unixa kaže, da je sistemske klice možno izvesti preko C ali C++ programa. Novejši Microsoft sistemi uporabljajo sistemske klice kot del Win32 **API** (application program interface), ki so uporabni za vse prevajalnike napisane za Microsoft Windows platformo.
Primer uporabe sistemskih klicev si poglejmo na primeru programa, ki kopira datoteko (prebere vsebino prve datoteke in zapiše prebrano vsebino v drugo datoteko).
Najprej mora program dobiti imena izvorne in ponorne datoteke. Imena datotek lahko podamo na več načinov. Lahko je to program, ki pokaže okno v katerem sprašuje po imenu izvorne ali ponorne datoteke (sistemski klic za prikaz okna, sistemski klic za branje s tipkovnice, lahko je datoteka na primer v Raziskovalcu, kjer jo grafično preko miške označimo (sistemski klici za delo z miško), ...Tak način zahteva nove sistemske klice. Ko dobimo imena datotek mora program odpreti izvorno datoteko in ustvariti ciljno datoteko

(nova sistemska klica). Poleg tega so možne tudi napake (datoteka ne obstaja, datoteka je zaščitena proti pisanju, ...), ki izpišejo uporabniku opozorila in opozorila se sprožijo preko zaporedja sistemskih klicev. Ko sta obe datoteki odprti naredimo zanko, ki bere vhodno datoteko (sistemski klic) in piše v ciljno datoteko (sistemski klic). Ko se kopiranje konča se datoteki zapreta (sistemska klica) in uporabniku sporočita, da je operacija uspešno zaključena (sistemski klic). Program se tako zaključi normalno (sistemski klic) brez napak.
Kot lahko vidimo programi izkoriščajo servise operacijskega sistema preko sistemskih klicev. Običajno sistemi izvedejo 1000 sistemskih klicev na sekundo.

Sistemske klice lahko razvrstimo v naslednje kategorije:

- **kontrola procesa** – program mora biti sposoben ustaviti izvajanje na normalen (*end*) ali abnormalen način (*abort*). Proces, ki med izvajanjem zahteva drug proces mora biti zmožen naložiti (*load*) in izvajati (*execute*) drug proces. Če kreiramo nov proces ali opravilo, moramo biti sposobni kontrolirati izvajanje procesa ali opravila. Kontrola zahteva zmožnost določiti in resetirati atribute procesa ali opravila (prioriteta opravila, maksimalni čas izvajanja, itd. - *get process attributes* in *set proces attributes*). Kreiran proces želimo tudi končati (*terminate proces*), če ni več potreben ali ne deluje pravilno. Če naredimo nov porces ali opravilo moramo počakati da konča svojo nalogo (*wait time*), bolj verjetno bomo čakali, da se zgodi nek dogodek (*wait event*). Opravilo ali proces signalizira kadar se je zgodil nek dogodek (*signal event*). Sistemske klice, ki koordinirajo istočasne procese bomo podrobno spoznali v poglavju o procesih.
- **operacije nad datotekami** – datoteke moramo ustvarjati (*create*) in brisati (*delete*). Ko datoteko ustvarimo jo moramo odpreti (*open*) in jo uporabiti – brati (*read*) in pisati (*write*) in končno datoteko moramo zapreti (*close*). Uporabljamo tudi atribute datotek, zato moramo atribute brati in spreminjati (*get file attribute* in *set file attribute*).
- **operacije z V/I napravami** – tekoči program lahko med izvajanjem potrebuje več virov sredstev za svoje delovanje (več pomnilnika, dostop do datotek, itd.) Če so viri dostopni jih lahko pridobi tekoči program in se kontrola programa prenese na razpoložljiv vir, drugače mora program čakati, da se vir sprosti. Kontrolo nad virom dobimo s zahtevo po napravi (*request*), ko dobimo nadzor nad njo jo uporabimo s sistemskimi klici za branje (*read*) in pisanje (*write*).
- **vzdrževanje informiranja** – veliko sistemskih klicev obstaja za prenos informacij med uporabniškim programom in operacijskim sistemom. Na primer sistemov ima sistemski klic, ki vrne trenutni datum (*date*) in čas (*time*) sistema. Drugi sistemski klici vrnejo druge informacije kot so število istočasnoh uporabnikov, verzija operacijskega sistema, velikost prostega pomnilnika, diska, itd..
- **komuniciranje** – poznamo dva modela komunikacije:
  - **potovanje sporočila** (message-passing) in
  - model **skupne rabe pomnilnika** (shared-memory).

  Pri metodi potovanja sporočila se mora komunikacija odpreti predenj se komunikacija začne. Ime druge strani s katero komunicira proces mora biti znana. Druga stran ja lahko na istem sistemu ali na oddaljenem sistemu povezanem preko omrežja. Vsak računalnik, ki je priključen v delujoče omrežje ima **omrežno ime** (host name – IP naslov). Enako ima proces **ime**, ki se prevede v identifikatorju preko katerega operacijski sistema proces prepozna. Sistemska klica *get hostid* in *get processid* naredita to pretvorbo. Proces naslovnik mora dovoliti da se komunikacija vzpostavi z *accept*

connection. Večina procesov, ki čaka na sprejem se imenujejo **daemons.** Daemoni so sistemski programi, ki izvedejo sistemski klic *wait for connection* in se zbudijo, ko je povezava vzpostaljena. Izvor komunikacije znan kot **odjemalec** (client) in sprejemnik deamon znan kot strežnik si izmenjujeta sporočila s sistemskima klicema *read message* in *write message*. Klic *close connection* konča komunikacijo.
V modelu skupne rabe pomnilnika procesi uporabljajo *map memory* sistemski klic. -s tem pridobijo dostop do področij v pomnilniku, ki jih lasti drug proces. Normalno operacijski sistem skuša preprečiti, da bi en proces dostopal do področja v pomnilniku, ki ga lasti drug proces. Skupna raba pomnilnika zahteva, da se oba procesa strinjata, da se odpravi zaščita dostopa. Procesa si izmenjujeta informacije tako, da pišeta in bereta podatke iz skupnega pomnilniškega prostora. Oblika podatkov in področja pomnilnika se določijo na nivoju procesa, operacijski sistem nima tukaj nobene kontrole. Procesi morajo tudi paziti, da ne pišejo na isto mesto istočasno.

3.4.1. Sistemski programi

Še en pogled na moderne operacijske sisteme je zbirka sistemskih programov. Če pogledamo o pogledu na logično strukturo računalniškega sistema vidimo, da je najnižje strojna oprema, operacijski sistem, sistemski programi in najvišje aplikacijski programi. sistemski programi skrbijo za udobno okolje v katerem se aplikacijski programi razvijajo in izvajajo. Nekateri so le vmesnik do sistemskih klicev, drugi pa bolj kompleksni. Razvrstimo jih lahko v naslednje kategorije:

- **upravljaje z datotekami** – programi kreirajo, brišejo, kopirajo, preimenujejo, tiskajo, ... datoteke in direktorije.
- **informacije o stanju sistema** – programi samo vprašajo sistem za datum, čas, velikost prostega pomnilnika ali disk, število uporabnikov, ...
- **spreminjanje datotek** – urejevalniki teksta ponavadi ustvarijo in spreminjajo vsebino datotek.
- **podpora programirnim jezikom** – prevajalniki, tolmači za razne programske jezike (C, C++, Java, Visual Basic, PERL, ...) so dani uporabniku poleg operacijskega sistema.
- **nalaganje in izvajanje programov** – ko je program enkrat preveden se mora naložiti v pomnilnik, da se izvaja. Sistem lahko priskrbi nalagalnike in urejevalnike prevedenih programov. Podprti so tudi sistemi za iskanje napak (debuger) na višjem nivoju ali strojnem nivoju.
- **komunikacije** – programi priskrbijo mehanizme za ustvarjanje virtualne povezave med procesi, uporabniki in računalniškimi sistemi. Omogočajo, da uporabnik pošlje sporočilo drugemu uporabniku in sporočilo se pojavi na ekranu, iskanje po spletnih straneh, pošiljanje elektronske pošte, prijavo na oddaljen sistem, prenašanje datotek iz enega sistema na drug sistem.

Kot dodatek sistemskim programom, večina operacijskih sistemov vsebuje programe, ki so uporabni pri reševanju običajnih problemov ali izvajanju določenih funkcij (internetni brskalniki, urejevalniki besedil in teksta, elektronske preglednice, bazne sisteme, igre, ...). Ti programi so poznani kot **sistemski pripomočki** ali **aplikacijski programi.**

3.5. Vprašanja za ponavljanje

1. Naštej glavne servise, ki jih nudi operacijski sistem.

2. Kakšne vmesnike lahko vsebuje operacijski sistem za komunikacijo uporabnik–operacijski sistem?

3. Kaj je sistemski klic?

4. S čim lahko pokličemo sistemski klic?

5. Kaj je sistemski program, kje v hierarhiji programov se nahaja?

4. Upravljanje procesov

Proces si lahko predstavljamo kot program v izvajanju. Proces med izvajanjem potrebuje določene vire sredstev (čas procesorja, spomin, datoteke, V/I naprave, ...), da bi izvedel nalogo.
Proces je enota dela v večini operacijskih sistemov. Tak sistem je sestavljen iz kolekcije procesov:

- **procesi operacijskega sistema** izvajajo sistemsko kodo in
- **uporabniški procesi** izvajajo uporabniško kodo

Vsi ti procesi se lahko izvajajo hkrati.

Tradicionalno je proces vseboval samo eno **nit** kontrole, ko se je izvajal, današnji operacijski sistemi pa podpirajo procese, ki vsebujejo več niti.

Operacijski sistem je odgovoren v povezavi z upravljanjem procesov za:

- ustvarjaje in brisanje sistemskih in uporabniških procesov
- razvrščanje procesov
- pripravo mehanizmov za sinhronizacijo, komunikacijo in odpravo smrtnih objemov za procese

4.1. Procesi

Proces je program v izvajanju. Proces je več kot kos programske kode (teksta v datoteki). Proces vsebuje programski del in podatkovni del.

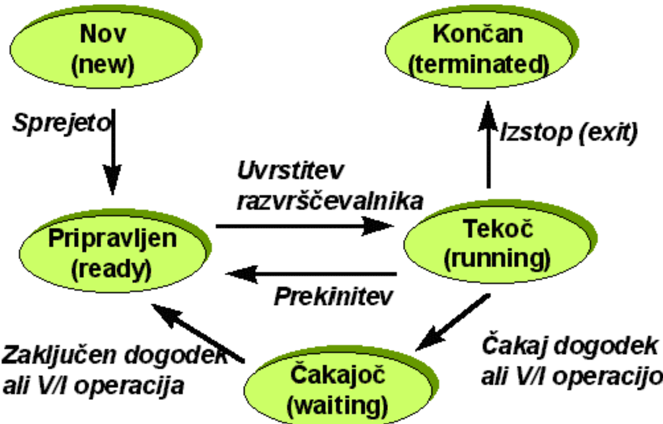
- **programsko kodo (text)**
- **trenutno stanje procesa**
    - **vsebina programskega števca**
    - **vsebine registrov** procesorja, ki opisuje trenutno aktivnost
- **sklad (stack)** procesa**,** ki vsebuje začasne podatke (parametre metode, vračilni naslov, lokalne spremenljivke)
- **podatkovni del (data)**, ki vsebuje globalne spremenljivke
- **dinamični pomnilnik (heap)**, ki se prilagaja potrebi procesa po pomnilniku med izvajanjem

Program je torej neaktivna-pasivna entiteta (npr. vsebina datoteke na disku), proces pa je aktivna entiteta s programskim števcem, ki kaže na naslednje navodilo ki naj se izvede.

4.1.1. Stanja procesa

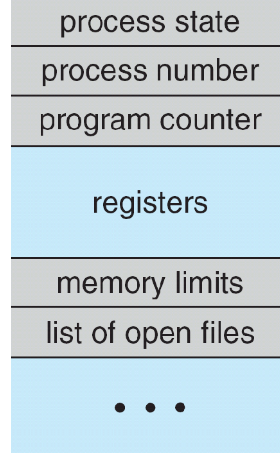
Ko se proces izvaja spreminja stanja. Proces lahko prehaja med 5 stanji:
- **Nov** (New) – Proces je v nastajanju
- **Tekoč** (Running) – Navodila v programski kodi se izvajajo preko **razvrščevalnika** v procesorju
- **Čakajoč** (Waiting) – Proces čaka na dogodek (zaključek V/I operacije, sprejem signala)
- **Pripravljen** (Ready) – Proces čaka, da ga dodeljevalnik (dispatcher) dodeli procesorju v izvajanje
- **Končan** (Terminated) – Proces je končal izvajanje.

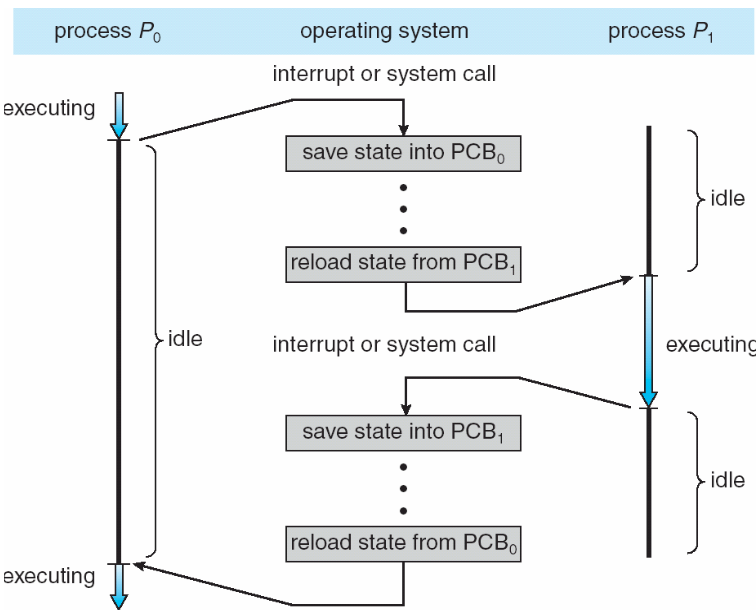
Samo en proces je lahko tekoč v nekem trenutku na enem procesorju, več procesov pa je v istem trenutku lahko čakajočih ali pripravljenih.



4.1.2. Kontrolni blok procesa (Process Control Block – PCB)

Vsak proces se predstavi operacijskem sistemu s kontrolnim blokom procesa. Kontrolni blok procesa vsebuje informacije, ki se nanašajo na določen proces:
- **Stanje procesa** - trenutno stanje v katerem se nahaja proces
- **Številka procesa - PID**
- **Programski števec** – kaže naslov navodila, ki se bo naslednje izvedlo
- **Registri procesorja** – varirajo v številu in tipu, odvisno od arhitekture sistema. Registri so lahko akumulatorji, indeksni registri, kazalniki na sklad, splošni registri, ... Vrednost programskega števca se mora začasno shraniti v kontrolni blok procesa, ko pride do prekinitve izvajanja procesa, da se izvajanje procesa, ko pride zopet na vrsto lahko nadaljuje tam kjer je končal izvajanje ob prekinitvi.
  Preklapljanje med procesoma kaže spodnje slika.



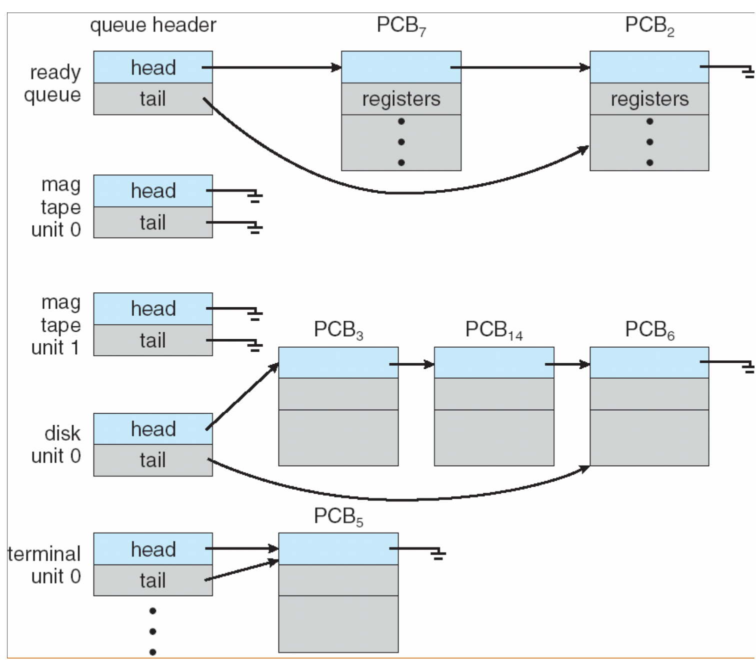


- **Informacije o razporejanju procesorja** – to so informacije o prioriteti procesa, kazalci na razporejanje vrste in ostali razporejevalni parametri.
- **Informacije o zasedenosti pomnilnika** – informacije o baznih in mejnih registrih, tabele strani, odvisno od pomnilniške arhitekture sistema.
- **Informacije o porabi** – poraba časa procesorja, časovne limite, številke procesov, ...
- **Stanje V/I naprav** – spisek V/I naprav, ki jih uporablja proces, spisek odprtih datotek, ...

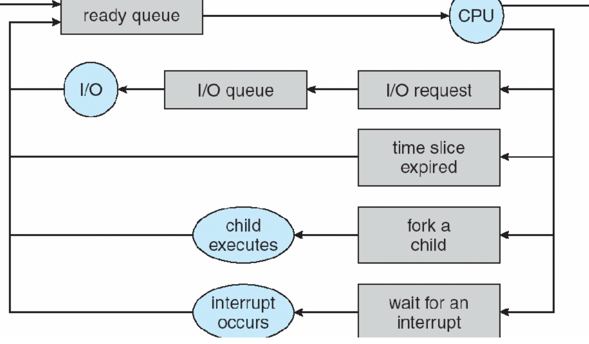
4.1.3. Razvrščanje procesov

Bistvo večopravilnega sistema je to, da lahko izvaja več procesov hkrati. Tako večopravilni sistem preklaplja delo procesorja med procesi tako pogosto, da uporabnik lahko izvaja več programov hkrati (ureja besedilo v urejevalniku besedil, posluša glasbo, ...).
Enoprocesorski sistem lahko naenkrat izvaja en proces imenovan tekoč proces. Če imamo več procesov morajo ostali počakati, da se delo procesorja, ki se ukvarja z nekim procesom sprosti in da procesorju razvrščevalnik procesov dodeli drug proces v obdelavo.

4.1.3.1. Razvrščevalna vrsta



Ko proces vstopi v sistem se postavi v **vrsto procesov**. V tej vrsti so vsi procesi, ki obstajajo v sistemu. Procesi, ki se nahajajo v glavnem pomnilniku in imajo stanje čakajoč ali pripravljen stojijo v tako imenovani **pripravljeni vrsti**. Pripravljena vrsta ima **glavo** (head), ki vsebuje kazalce na prvi PCB in zadnji PCB v pripravljeni vrsti.
Ko je proces dodeljen procesorju, se izvaja in nato se lahko konča, se prekine preko prekinitve ali pa čaka, da se nek dogodek konča (npr. V/I zahteva – npr. branje z diska). Ker ima sistem veliko procesov se lahko zgodi, da na isto V/I napravo (npr. disk) čaka več procesov. Zato operacijski sistem vsebuje poleg pripravljene vrste tudi **vrsto za vsako V/I napravo** (npr. vrsto za disk, terminal, ...). V vrsti V/I naprave čakajo procesi, ki čakajo na V/I operacijo na tej napravi.
Na spodnji sliki vidimo **diagram vrst**, ki nadzorno prikazuje razvrščanje procesov. Vsak pravokotnik predstavlja vrsto. Vidimo dva tipa vrst – **pripravljeno vrsto** in **vrste za V/I naprave**. Krogi predstavljajo sredstvo, ki servira procesu v vrsti. Puščice pa kažejo pretok procesov v sistemu.



Nov proces je vedno postavljen v pripravljeno vrsto in čaka, da pride na vrsto za izvajanje v procesorju. Ko se proces izvaja oz. je v tekočem stanju se lahko zgodi več dogodkov:

- proces lahko zahteva V/I operacijo in je zato postavljen v vrsto V/I naprav
- proces med izvajanjem lahko ustvari podproces in čaka, da se podproces konča izvajati.
- izvajanje procesa lahko proti njegovi volji ustavi prekinitev in proces je potisnjen nazaj v pripravljeno vrsto.

Posamezen proces v svojem življenju potuje iz ene vrste v drugo, odvisno od dogodkov. Ko proces konča izvajanje se izbriše iz vseh vrst.
Procesor ponavadi porabi 10 % časa za opravljanje razvrščanja procesov.

4.1.3.2. Razvrščevalnik

Proces v njegovem življenju migriria med različnimi razvrščevalnimi vrstami. Operacijski sistem mora na nek način izbirati procese iz čakalne vrste. Izbiro procesa iz čakalne vrste določa **razvrščevalnik**.
Poznamo tri vrste razvrščevalnikov:

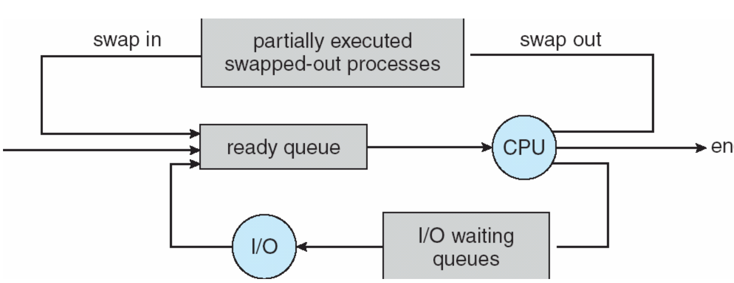
- v paketnih sistemih so lahko procesi, ki se niso izvajali določen čas začasno shranjeni na

napravah za shranjevanje podatkov (tipično virtualni pomnilnik na magnetnem disku). **Dolgoročni razvrščevalnik** ali **razvrščevalnik opravil** izbere proces na magnetnem disku in ga naloži v pomnilnik, kjer se bo izvajal.
- **Kratkoročni razvrščevalnik** ali **razvrščevalnik na procesorju** pa izbere proces v glavnem pomnilniku izmed procesov, ki so pripravljeni vrsti.

Primarna razlika je v frekvenci izvajanja razvrščevalnikov.
Kratkoročni mora izbirati procese zelo hitro. Navadno je potreba po kratkoročnem razvrščevalniku vsakih 100 ms. Zaradi hitrega izbiranja mora biti sam razvrščevalnik hiter. Če na primer porabi sam razvrščevalnik 10 ms za izbiranje procesa v 100 ms intervalu pomeni, da je 9% (10/(10+100)) časa namenjenega razvrščanju na procesorju. Procesor torej porabi 9 % časa za opravljanje razvrščanja.
Na drugi strani se lahko dolgoročni razvrščevalnik izvede vsakih nekaj minut. Dolgoročni razvrščevalnik zato kontrolira **stopnjo multiprogramiranja** – število procesov v pomnilniku. Če je stopnja multiprogramiranja stabilna ali konstantna bo število procesov, ki pride v pomnilnik enaka številu procesov, ki zapusti pomnilnik. Dolgoročni razvrščevalnik se lahko uporablja kadar proces zapusti sistem (verjetnost, da se ga bo zopet potrebovalo).

V osnovi lahko večino procesov razdelimo na **procese dodeljene procesorju** in **procese dodeljene V/I napravam**. Procesi dodeljeni V/I napravam porabijo več svojega časa za V/I operacije kot računanje, procesi dodeljeni procesorju pa generirajo V/I zahteve bolj pogosto, kjer porabijo več svojega časa za računanje, kot pa procesi dodeljeni V/I napravam.
Dolgoročni razvrščevalnik mora izbrati dobro kombinacijo obeh vrst procesov, kajti če izbere procese dodeljene V/I napravam bo pripravljena vrsta prazna in kratkoročni razvrščevalnik ne bo imel dela, če pa izbere procese dodeljene procesorju bo V/I vrsta prazna in V/I naprave ne bodo imele dela – sistem bo **neuravnotežen**. Uravnotežen sistem z najboljšo učinkovitostjo mora imeti pravilno izbiro procesov, ki so dodeljeni procesorju in procesov, ki so dodeljeni V/I napravam. Za ta namen so nekateri večopravilni operacijski sistemi uvedli srednjeročno stopnjo razvrščanja.

- **Srednjeročni razvrščevalnik** premakne delno izveden proces iz pomnilnika (in registrov procesorja) in s tem zmanjša stopnjo multiprogramiranja. Kasneje se proces lahko zopet postavi v pripravljeno vrsto in proces se lahko nadaljuje kjer se je končal. Tak način se imenuje tudi **izmenjalni način** (swapping). Proces je začasno postavljen na sekundarni medij za shranjevanje podatkov (npr. magnetni disk) in nazaj postavljen v glavni pomnilnik preko srednjeročnega razvrščevalnika. To je potrebno kadar hočemo izboljšati izbiro med procesi dodeljenimi procesorju in procesi dodeljenimi V/I napravam, ter ko v glavnem pomnilniku zmanjkuje prostora in je potreba po sprostitvi glavnega pomnilnika – navidezni pomnilnik, izmenjalni ali "swap" razdelek na disku! Več o tem v poglavju Upravljanje shranjevanja.



4.1.3.3. Menjava okolja (Context switch)

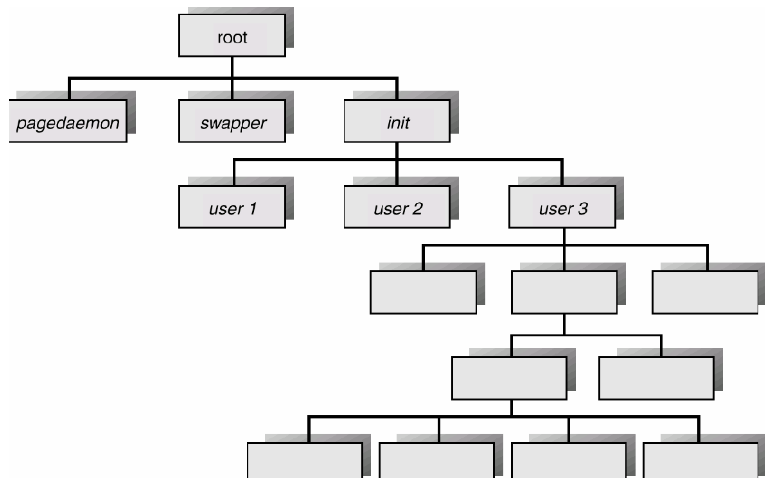
Menjava dela procesorja na drug proces zahteva shranjevanje stanja procesa, ki zapušča procesor in nalaganje stanja procesa, ki prihaja v izvajanje v procesor. Naloga je znana pod imenom **menjava okolja**. V okolje procesa spada kontrolni blok procesa (PCB) (stanja registrov v procesorju, stanje procesa in informacije o upravljanju pomnilnika). Ko se zgodi menjava okolja, jedro shrani PCB procesa, ki zapušča procesor in naloži PCB procesa, ki prihaja v izvajanje v procesor. Čas menjave okolja je **jalov čas** (overhead), čas ko za nas ne počne nič koristnega.
Jalov čas je odvisen od hitrosti pomnilnika, števila registrov, ki se morajo shraniti in od posebnega navodila (navodilo, ki shrani in naloži PCB). Tipične hitrosti so od 1 do 1000 µs.

4.1.4. Operacije nad procesi

Operacijski sistem mora poskrbeti mehanizem za ustvarjanje in brisanje procesov v sistemu.

4.1.4.1. Ustvarjanje procesov

Proces lahko med njegovim izvajanjem ustvari nov proces preko sistemskega klica "create-process". Ustvarjen proces se imenuje **otroški proces**, proces, ki ga je ustvaril pa **starševski proces**. Vsak proces lahko naredi nov proces in procese lahko prikažemo v drevesni strukturi.



4.1.4.2. Končevanje procesov

Proces se konča, ko se izvede zadnje stavek v programski kodi. Proces vpraša operacijski sistem, da ga pobriše s sistemskim klicem "exit". Na tej točki lahko proces vrne podatke (kot rezultate procesa) svojemu starševskemu procesu s sistemskim klicem "wait". Vsa sredstva (fizični in virtualni pomnilnik, odprte datoteke, registri V/I naprav), ki jih je izbrisan proces uporabljal se sprostijo preko operacijskega sistema.
Končevanje procesov se lahko zgodi pod določenimi pogoji. Samo starševski proces lahko konča otroški proces preko sistemskega klica "abort". Če nebi bilo tako, bi lahko proces enega uporabnika končal proces drugega uporabnika.
Starševski proces lahko konča svoje otroške procese ker:

- je otroški proces presegel uporabo sredstva, ki so mu bila dodeljena. To zahteva od starševskega procesa, da ima mehanizem o svojih otroških procesih, kjer preverja

stanje otroškega procesa.

- je naloga, ki je bila dodeljena otroškemu procesu, ni več potrebna
- starševski proces je v fazi končevanja in operacijski sistem ne dovoli, da bi se otroški procesi izvajali brez njihovih starševskih procesov. Če se proces konča (normalno ali nenormalno) se morajo končati tudi vsi otroški procesi – operacijski sistem **kaskadno končuje otroške procese**.

4.1.5. Sodelovanje procesov

Vzporedno ali istočasno izvajanje procesov v lahko **sodelujoče** ali **neodvisno**.
Neodvisen proces je tisti, ki ne more vplivati na druge procese, oz. drugi procesi ne morejo vplivati nanj. Vsak proces, ki ne daje svojih podatkov v skupno rabo je neodvisen proces.
Sodelujoč proces je tisti, ki lahko vpliva na druge procese, oz. drugi procesi lahko vplivajo nanj. Vsak proces, ki si izmenjuje podatke z drugimi procesi je sodelujoč proces.
Priskrbeti moramo okolje v katerem bomo omogočili sodelovanje med procesi zaradi:

- **skupne rabe informacij** – več uporabnikov uporablja isto informacijo (npr. isto datoteko)
- **pospešitev računanja** – če želimo, da se bo neka naloga izvedla hitreje jo moramo razdeliti na podnaloge
- **modularnosti** – razdelitev funkcij sistema v več ločenih procesov ali niti

Sodelovanje procesov poteka preko skupnega naslovnega prostora v pomnilniku, kjer mora uporaba skupnega prostora biti eksplicitno napisana s strani programerja – v programski kodi. Pri tem se pojavi problem pisca in bralca iz skupnega prostora, kjer mora biti pisanje in branje v naslovni prostor usklajeno.
Kot ilustracijo sodelovanja procesov lahko opišemo problem **proizvajalec-porabnik**.
Proizvajalec proizvaja informacijo, ki jo uporabnik uporablja. Informacija, ki jo proizvajalec proizvede se shrani na skupni prostor v pomnilniku. Če je pomnilniški prostor omejen moramo paziti, da porabnik počaka, da se pomnilnik napolni z informacijami, ki jih je proizvedel proizvajalec. Na drugi strani pa mora proizvajalec počakati, ko je pomnilnik poln, da porabnik porabi informacije. Skratka, delovanje proizvajalca in porabnika je treba uskladiti – **sinhronizirati**.
Primer rešitve problema proizvajalec-porabnik lahko prikažemo na spodnjem primeru.
Procesa proizvajalca in porabnika si delita naslednje spremenljivke:

```
#definiraj VELIKOST_BUFFERJA 10
typedef struct {
...
}item ;
item bufer[VELIKOST_BUFERJA];
int noter=0;      #int definira celoštevilčni tip globalne spremenljivke noter
                  #in dodeli začetno vrednost 0
int ven=0;
```

Imamo skupni pomnilniški prostor – "buffer", ki ga predstavimo kot krožno polje z dvema logičnima kazalcema (`noter, ven`).
Spremenljivka `noter` kaže na prvo naslednjo prazno lokacijo v "bufferju", spremenljivka `ven` pa kaže na prvo polno lokacijo v "bufferju". Sledi, da je "buffer" poln, ko velja:
`noter==ven`
in prazen ko velja:
`((noter+1)%VELIKOST_BAFERJA)==ven`

Sledijo kode programa za proizvajalca in porabnika.
Proces proizvajalca ima lokalno spremenljivko `proizvedi_naslednjega`, kjer je shranjena informacija, ki jo proizvajalec proizvede in shrani na določeno mesto v "bufferju".

Koda programa za proizvajalca:

```
while (1) {
      /*proizvedi informacijo v proizvedi_naslednjega */
      while (((noter+1)%VELIKOST_BAFERJA)==ven)
      ; /*ne naredi ničesar*/
      bafer[noter]= proizvedi_naslednjega;
      noter=(noter+1)%VELIKOST_BAFERJA;
}
```

Proces porabnika pa ima lokalno spremenljivko `porabi_naslednjega`, kjer je shranjena informacija, ki jo je koda programa porabnika prebrala na prvem polnem mestu v baferju.

```
while (1) {
      while (noter==ven)
      ; /*ne naredi ničesar*/
      porabi_naslednjega = bafer[ven];
      ven=(ven+1)%VELIKOST_BAFERJA;
      /* porabi informacijo v porabi_naslednjega */
}
```

Tak način dovoli največ `VELIKOST_BUFFERJA-1` poln buffer, v bufferju ostaja v najboljšem primeru eno prazno mesto.

Več o tem v poglavju Sinhronizacija procesov.

4.1.6. Medprocesno komuniciranje

Sodelovanje med procesi lahko poteka tudi preko **medprocesne komunikacije** (Interprocess Communication IPC). IPC zagotavlja mehanizem, da procesi med seboj komunicirajo brez uporabe skupnega naslovnega prostora v pomnilniku. IPC je predvsem uporaben v porazdeljenih sistemih, kjer komunikacijski proces domuje na dveh različnih sistemih povezanih z mrežo (npr. chat program – pogovor med uporabniki v realnem času). IPC najbolje predstavlja sistem prenosa sporočil.

4.1.6.1. Sistem prenosa sporočil

Komunikacija med uporabniškimi procesi poteka preko sporočil. IPC komunikacija priskrbi vsaj dve operaciji: **pošlji sporočilo** (send message) in **sprejmi sporočilo** (receive message).
Če procesa A in B želita komunicirati morata obojestransko sprejemati in pošiljati sporočila eden drugemu. Med njima mora obstajati **komunikacijska povezava**.

4.1.6.2. Imenovanje

Procesi, ki bi radi medsebojno komunicirali morajo na nek način vedeti kdo je kdo.

Uporabljajo lahko direktno ali indirektno komuniciranje.

4.1.6.2.1. Direktno komuniciranje

Zahteva se eksplicitno imenovanje sprejemnika in pošiljatelja. V tem načinu sta "send" in "receive" operacija definirani takole:
- send(A, message) – pošlji sporočilo procesu A
- receive(B, message) – sprejmi sporočilo od procesa B

Komunikacijska povezava ima naslednje lastnosti:
- povezava se avtomatsko vzpostavi, ko obstaja par procesov, ki želita komunicirati. Procesa potrebujeta samo ime drugega procesa za komunikacijo
- povezava je vzpostavljena z natančno dvema procesoma
- samo ena povezava obstaja med dvema procesoma

4.1.6.2.2. Indirektno komuniciranje

Sporočila so tule poslana in sprejeta preko poštnih predalov ali vrat.
- send(A, message) – pošlji sporočilo poštnemu predalu A
- receive(B, message) – sprejmi sporočilo iz poštnega predala A

4.1.6.3. Primer: Windows XP

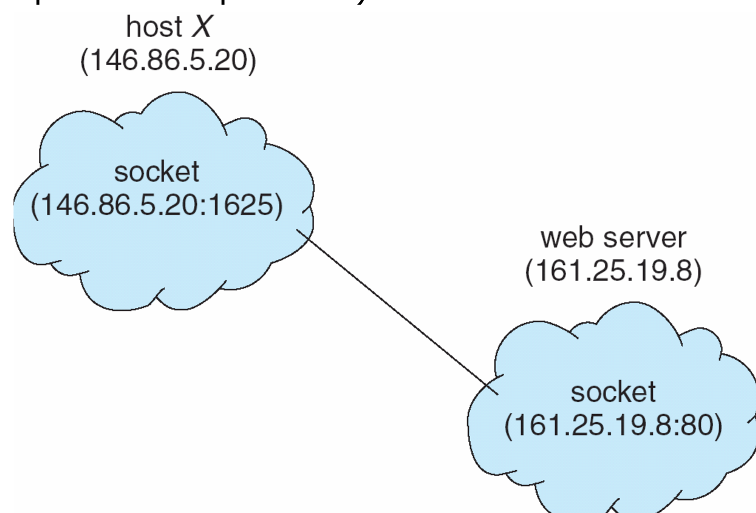
Windows XP zagotavlja podporo za več okolij za delovanje – podsisteme (subsystems) s katerimi aplikacijski programi komunicirajo med sistemi preko sistema prenosa sporočil. Aplikacijski program predstavlja odjemalca za Windows XP strežniški podsistem.
Prenos sporočil v Windows XP operacijskem sistemu poteka preko LPC - "local procedure call". To omogoča komunikacijo med dvema procesoma, ki sta na istem sistemu. Vsak odjemalec, ki pokliče podsistem mora vzpostaviti komunikacijski kanal, ki deluje na principu objekta vrat - **indirektnem komuniciranju** in ni nikoli podedovan.

4.1.7. Komunikacija odjemalec-strežnik

Omislimo si uporabnika, ki bi želel ugotoviti koliko vrstic, besed in znakov vsebuje datoteka na strežniku A. Zahtevo bi prevzel strežnik A, ki bi odprl datoteko, opravil določene operacije in poslal rezultate nazaj uporabniku.

4.1.7.1. Vtiči (sockets)

Vtič je definiran kot končna točka komunikacije. Par procesov, ki komunicira preko omrežja zaposli par vtičev – za vsakega enega. Vtič je sestavljen iz IP številke v povezavi s številko vrat. Vtiči uporabljajo odjemalec-strežnik arhitekturo. Strežnik čaka na zahteve odjemalca tako, da posluša na določenih vratih (npr. http strežnik posluša na vratih 80, ftp strežnik posluša na vratih 21, ...). Vrata z nižjo številko od 1024 so standardizirana vrata in jih lahko uporabljamo za implementacijo v določene servise.
Ko proces na strani odjemalca zahteva komunikacijo s strežnikom, mu strežnik dodeli poljubno številko vrat, ki je običajno večja od 1024. Tako se vzpostavi komunikacijski kanal:
npr. 146.86.5.20:1690 (vtičnica odjemalca) – 161.25.19.8:80 (vtičnica strežnika na vratih 80 - vrata na katerih posluša http strežnik http zahteve).
Če bi nek drug proces hotel vzpostaviti povezavo z istim strežnikom, bi se vzpostavila povezava, kjer bi vtič na strani strežnika ostal nespremenjen, na strani odjemalca pa bi bila številka vrat večja kot 1024 in različna od 1625.

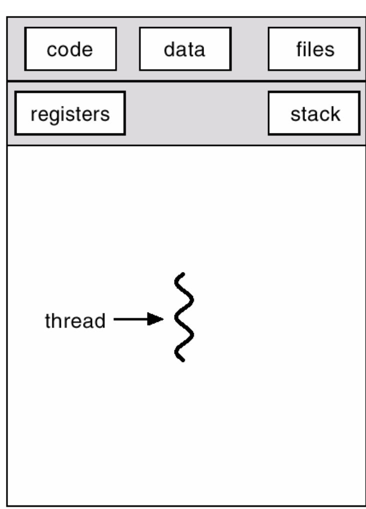


4.2. Niti

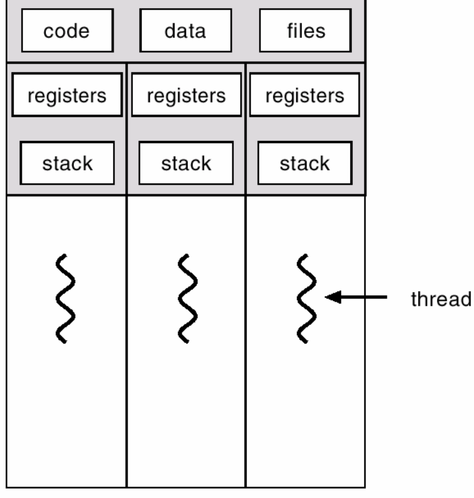
V prejšnjem poglavju smo obravnavali proces z eno nitjo kontrole. Večina operacijskih sistemov danes omogoča izvajanje procesov z več nitmi kontrole. Proces je lahko **enoniten** ali **večniten**.
Posamezna nit vsebuje:

- identifikacijo niti (ID)
- programski števec
- nekaj registrov
- sklad.

Z ostalimi nitmi si deli programsko kodo, podatke in druge vire sistema npr. datoteke. Če ima proces več niti lahko naredi več opravil naenkrat!
Veliko



single-threaded



multithreaded

programskih aplikacij, ki tečejo na modernih operacijskih sistemih je **večnitnih**. Običajno je aplikacija, ko teče ločen proces z več nitmi kontrole.
Kot primer lahko navedemo spletni brskalnik, ki z eno nitjo kontrolira prikaz spletne vsebine (tekst, slike, ...) na zaslon, z drugo nitjo pa kontrolira sprejemanje in pošiljanje mrežnih paketov. Urejevalnik besedil je ločen proces, ki ima eno nit, ki kontrolira prikaz besedila na zaslon, drugo nit, ki sprejema ukaze uporabnika in tretjo nit, ki recimo izvaja črkovanje na vtipkanem besedilu. Ponavadi se od določene aplikacije zahteva, da opravi več enakih nalog. Kot primer vzemimo spletni strežnik, ki servira zahteve odjemalcev. Ko se vzpostavi komunikacijski kanal strežnik kreira ločeno nit, ki servira zahtevo. Spletni strežnik ima lahko več deset istočasnih zahtev. Če bi strežnik deloval kot enonitni proces, bi lahko serviral samo enemu odjemalcu naenkrat, ostali bi morali čakati.

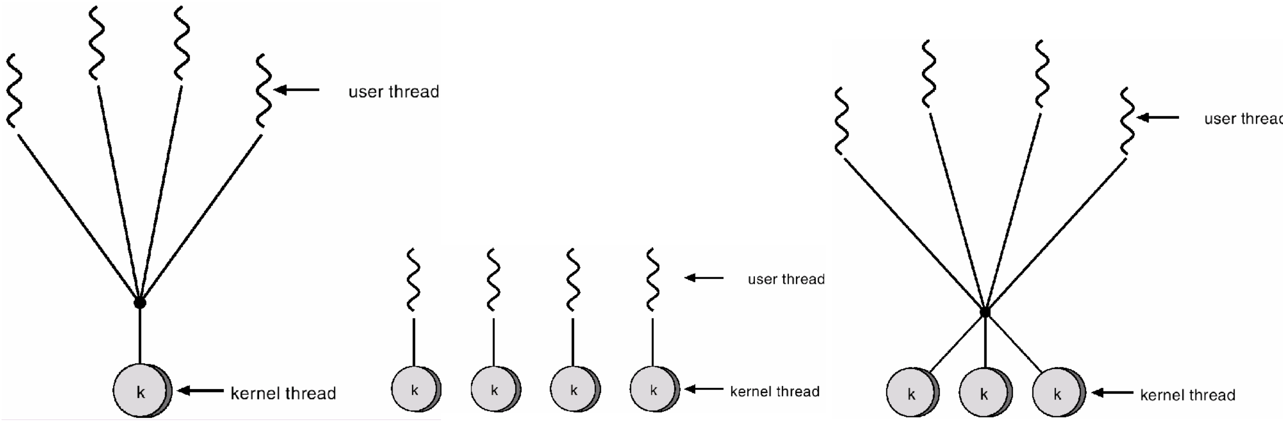
4.2.1. Uporabniške niti in niti jedra

Podpora večnitnemu procesu je lahko na **uporabniškem nivoju** in na **nivoju jedra**. Tako ločimo **uporabniške niti** in **niti jedra**.

- uporabniške niti so podprte nad jedrom na uporabniškem nivoju preko **nitnih knjižnic**, ki priskrbijo podporo za ustvarjanje, razvrščanje in upravljanje uporabniških niti. Nitne knjižnice za uporabniške niti so Pthreads (od standarda POSIX), C-threads (Mach) in UI-threads (Solaris 2).

- niti jedra so podprte direktno preko operacijskega sistema. Jedro priskrbi ustvarjanje, razvrščanje in upravljanje z nitmi na nivoju jedra. Ponavadi je ustvarjanje niti jedra počasnejše od ustvarjanja niti na uporabniškem nivoju. Večina današnjih operacijskih sistemov (Windows NT, Windows 2000, Solaris 2, BeOS, Unix, ...) podpira niti jedra.

4.2.2. Večnitni modeli



4.2.2.1. Več na ena model

Združuje več uporabniških niti v eno nit jedra. Upravljanje niti se izvaja na uporabniškem nivoju, zato je efektivno. Slabost v tem modelu pa je ta, da če ena nit pošlje sistemski klic (npr. "wait"), se celoten proces ustavi in čaka. Knjižnice niti za Solaris 2 uporabljajo ta model.

4.2.2.2. Ena na ena model

Združuje eno uporabniško nit z natančno eno nitjo jedra. Ko pride do sistemskega klica (npr. "wait"), se lahko ostale niti izvajajo. Edina slabost je, da vsaka uporabniška nit ustvari eno nit v jedru in niti v jedru se ustvarijo počasneje. Tak model uporabljajo Windows NT, Windows 2000 in OS/2 sistemi.

4.2.2.3. Več na več model

Tule prihaja do multipleksiranja več uporabniških niti z več nitmi jedra. Število niti jedra je odvisno od aplikacije ali celo sistema. Solaris 2, IRIX, HP-UX in Tru64 Unix podpirajo ta model.

4.3. Vprašanja za ponavljanje

1. Kaj je proces in kaj ga sestavlja?

2. Naštej stanja v katerih se lahko proces nahaja. V kakšnem stanju je lahko samo en proces?

3. Preko koga se proces predstavi operacijskemu sistemu?

4. V katerih vrstah lahko čakajo procesi?

5. Razloži razliko med kratkoročnim, dolgoročnim in srednjeročnim razvrščanjem.

6. Kaj je menjava okolja (context switch) in kaj je slabost menjave okolja?

7. Kaj počne jedro OS, ko se menjava okolje?

8. Kaj je sodelujoč proces?

9. Kakšne modele medprocesne komunikacije poznaš?

10. Kaj je nit (thread)?

11. Naštej prednosti kreiranja niti namesto procesa?

5. Razvrščanje na procesorju

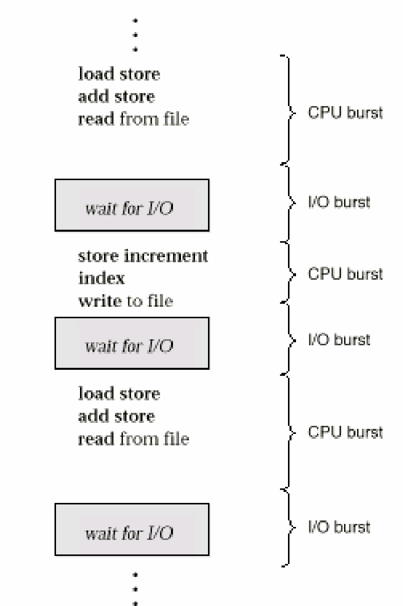
Razvrščanje na procesorju je osnova večopravilnemu delovanju operacijskega sistema. Ko procesor preklaplja med procesi, lahko operacijski sistema naredi sistem bolj produktiven.

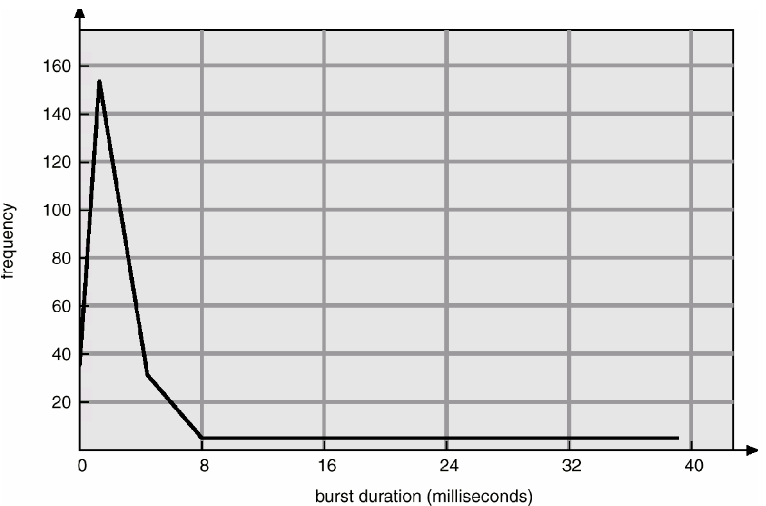
5.1. Osnovni koncepti

Cilj multiprogramiranja je imeti več procesov, ki se neprestano izvajajo. V enoprocesorskih sistemih se lahko na procesorju izvaja naenkrat samo en proces. Katerikoli drugi proces mora čakati, da se procesor sprosti, izvede se razvrščanje in procesor dobi v izvajanje drug proces.
Ideja multiprogramiranja je dokaj enostavna. Proces se izvaja dokler, tipično, mora čakati na V/I operacijo. V enostavnih sistemih bi procesor miroval in čas mirovanja bi bil izgubljen. Z večopravilnostjo se čas mirovanja izkoristi bolj produktivno.
Pri večopravilnosti imamo več procesov v pomnilniku naenkrat. Ko mora proces čakati, tipično, na V/I operacijo, operacijski sistem procesorju po načelu razvrščanja dodeli drug proces.
Razvrščanje je torej osnovna funkcija operacijskega sistema. Skoraj vsa računalniška sredstva so razvrščena predenj jih uporabimo. Procesor spada med primarna sredstva sistema. Tako je razvrščanje na procesorju osnovna naloga operacijskega sistema.

5.1.1. V/I delovni cikel procesorja

Uspešnost razvrščanja na procesorju je odvisna od naslednjih lastnosti procesa:
Izvajanje procesa se sestoji iz **cikla izvajanja na procesorju** in **cikla čakanja na V/I napravo**. Procesi preklapljajo med tema cikloma. Izvajanje procesa se začne z ciklom izvajanja na procesorju, sledi mu cikel čakanja na V/I napravo, nato naslednji cikel izvajanja na procesorju, itd.
Trajanje ciklov so izmerili in naredili krivuljo, ki predstavlja frekvenco preklapljanja ciklov glede na dolžino cikla. Na sliki vidimo visoko frekvenco preklapljanja ciklov pri kratkih ciklih in majhno frekvenco preklapljanja pri dolgih ciklih.
Pri ciklih, ki so dolgi približno 2-3 ms je frekvenca preklapljanja najvišja. Tole nam pomaga izbrati primerno razvrščanje na procesorju.


load store
add store
read from file
CPU burst
wait for I/O
I/O burst
store increment
index
write to file
CPU burst
wait for I/O
I/O burst
load store
add store
read from file
CPU burst
wait for I/O
I/O burst


frequency
160
140
120
100
80
60
40
20
0
8
16
24
32
40
burst duration (milliseconds)

5.1.2. CPE razvrščevalnik

Ko ostane procesor brez dela mora operacijski sistem izbrati enega izmed procesov v pripravljeni vrsti. Za razvrščanje poskrbi kratkoročni razvrščevalnik (CPE razvrščevalnik). Razvrščevalnik izbere med procesi, ki so v pripravljeni vrsti in dodeli procesor enemu od njih.

5.1.2.1. Razvrščanje s ali brez prekinjanja ((non)preemptive scheduling)

Odločitve o razvrščanju na procesorju se lahko pojavijo v 4 situacijah:

1. Ko proces preklopi iz tekočega v čakalno stanje (npr. V/I zahteva, prošnja za čakanje, da se konča podproces (otroški proces).
2. Ko proces preklopi iz tekočega v pripravljeno stanje (npr. ko se zgodi prekinitev).
3. Ko proces preklopi iz čakalnega stanja v pripravljeno stanje (npr. konec V/I operacije – prekinitev preko V/I naprave).
4. Ko se proces konča.

Ko razvrščanje poteka v primerih 1 in 4 govorimo o **razvrščanju brez prekinjanja**, ko proces **sam** zahteva drugo stanje, v primerih 3 in 4 pa govorimo o **razvrščanju s prekinjanjem**, kjer proces prekine zunanji vpliv (npr. prekinitev).
Pri razvrščanju brez prekinjanja , ko enkrat proces zaposli procesor, se proces izvaja dokler ne pride do končanja procesa ali do prehoda v čakalno stanje. Taka metoda razvrščanja je uporabljena v operacijskem sistemu Windows 3.1. Ta način razvrščanja uporabljamo tam, kjer nimamo potrebne strojne opreme (npr. časovnik) za uporabo razvrščanja s prekinjanjem.
Windows 95 uveljavlja razvrščanje s prekinjanjem, ki se nadaljuje v naslednice Windows 95 sistema.
Prekinjanje se lahko zgodi naključno. Zato je slabost razvrščanja s prekinjanjem v tem, da lahko pride do nekonsistentnosti podatkov pri sodelujočih procesih. Npr. proces med izvajanjem piše v skupni pomnilniški prostor, nakar je na sredini pisanja prekinjen. Ob prekinitvi dodeli razvrščevalnik procesor drugemu procesu, ki npr. bere iz skupnega pomnilniškega prostora. Prebrani podatki so v tem primeru nepravilni oz. nekonsistentni.

5.1.2.2. Dodeljevalnik (dispatcher)

Še ena komponenta razvrščanja na procesorju je dodeljevalnik. To je komponenta, ki daje kontrolo procesorju pri izbiri procesa preko kratkoročnega razvrščevalnika.
Dodeljevalnik vsebuje:

- Menjavo okolja
- Preklop v uporabniški način dela
- Skok na lokacijo v pomnilniku, kjer se lahko uporabniški program ponovno zažene.

Dodeljevalnik mora biti kar najbolj hiter, saj se požene ob vsakem preklopu procesa. Čas, ki ga dodeljevalnik porabi, da ustavi en proces in zažene drug proces imenujemo **dodeljevalna latenca**.

5.1.3. Kriteriji razvrščanja

Različni razvrščevalni algoritmi imajo različne lastnosti, kateri proces bodo izbrali in dodelili procesorju v izvajanje. Kateri razvrščevalni algoritem izbrati v določeni situaciji, moramo preučiti lastnosti posameznega razvrščevalnega algoritma.
Kriterij izbire razvrščevalnega algoritma zajemajo:

- **izkoriščenost procesorja** – procesor hočemo čimbolj zaposliti. Zaposlitev procesorja je lahko od 0 do 100 %. V realnem sistemu bi se moral gibati okoli 40 % (za lahko obremenjen sistem) in okoli 90 % (za močno obremenjen sistem),
- **propustnost sistema** (throughput) – (število končanih procesov/časovno enoto) mora biti čim večje. Za dolge procese je količnik 1 proces/h, za kratke procese pa količnik 10 procesov/s,
- **čas obdelave** (turaround time) – čas, ko se posamezen proces prvič začne izvajati do trenutka, ko je proces zaključil svojo nalogo,
- **čakalni čas** (waiting time) – čas, ki ga proces porabi, ko čaka v pripravljeni vrsti. **Čakalni čas** je vsota časov porabljenih za čakanje procesa v pripravljeni vrsti.
- **odzivni čas** (response time) – v interaktivnem sistemu, čas obdelave ni primeren kriterij. Zato se raje uporablja **odzivni čas**, ki je čas od kreiranja procesa do prve V/I aktivnosti procesa.

Nekatera merila so med seboj povezana (tudi obratno sorazmerna), zanimajo nas ekstremi – min, max, povprečja…
Maksimizirati želimo izkoriščenost procesorja in propustnost sistema, minimizirati pa čas obdelave, čakalni čas in odzivni čas. V večini primerov optimiziramo povprečne vrednosti. Vendar v nekaterih primerih raje optimiziramo minimalne in maksimalne vrednosti kot povprečne.
Pri interaktivnih sistemih je bolj pomembno minimizirati varianco odzivnega časa kot povprečen odzivni čas, saj je sistem s sprejemljivim in predvidljivim odzivnim časom bolj zaželen kot sistem, ki je hitrejši v povprečju, vendar zelo spremenljiv.
Minimizacija oz. maksimizacija, ki jo želimo izvesti, je splošen optimizacijski problem. Obvladljivi so le posebni problemi, npr. povprečni časi, maksimizacija izkoriščenosti procesorja,... Ostali optimizacijski problemi pa so večinoma neobvladljivi, NP-polni (nedeterministično polinomski) – problemi, za katere v praksi ne obstaja algoritem. Taki problemi se rešujejo hevristično: pregleda se razmeroma majhna podmnožica rešitev neobvladljivega problema in "najboljša" od teh rešitev se vzame kot rešitev problema, za katero nimamo dokaza, da je blizu optimalni, čeprav običajno je.

5.1.4. Algoritmi razvrščanja na procesorju

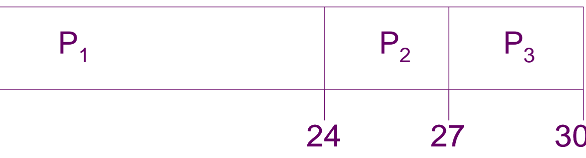
Animacije: http://gaia.ecs.csus.edu/~zhangd/oscal/pscheduling.html
Razvrščanje na procesorju se ukvarja s problemom odločitve, kateri proces v pripravljeni vrsti dodeliti procesorju v izvajanje. Opisali bomo nekaj algoritmov razvrščanja na procesorju.

5.1.4.1. “Prvi pride, prvi melje” algoritem razvrščanja (First-Come, First-Served – FCFS)

To je najenostavnejši algoritem razvrščanja na procesorju. Po tem algoritmu razvrščevalnik postreže tisti proces, ki to prvi zahteva. Ko proces vstopi v pripravljeno vrsto, se njegov PCB postavi na zadnje mesto v vrsti. Ko je procesor prost se mu dodoeli prvi proces v vrsti. Tekoč proces se nato izloči iz čakalne vrste pripravljenih procesov. Povprečni čakalni čas (waiting time) je dokaj dolg.
Kot primer si vzemimo nabor procesov, ki pridejo ob času 0, s ciklom na procesorju podan v ms.

Proces	Cikel
P_1	24
P_2	3
P_3	3

Če procesi pridejo v zaporedju P_1, P_2, P_3 dobimo naslednjo sliko.



Čakalni čas je v takem primeru 0 ms za P_1, 24 ms za P_2 in 27 ms za P_3.
Povprečni čakalni čas je $(P_1+P_2+P_3)/3$ = (0 ms+24 ms+27 ms) = **17 ms**.

Če procesi pridejo v zaporedju P2, P3, P1 dobimo naslednjo sliko:

P_2 P_3 P_1
0 3 6 30

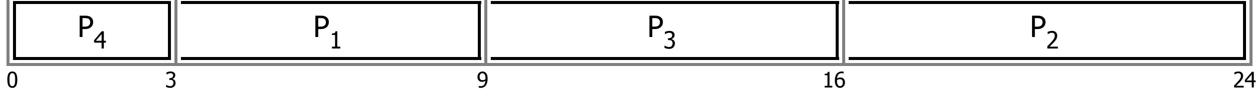
Povprečen čakalni čas je v tem primeru (0 ms+3 ms+6 ms)/3 = **3 ms**.
Vidimo, da je povprečen čakalni čas zelo odvisen od zaporedja procesov, ki vstopajo v čakalno vrsto in cikla na procesorju, ki ga posamezen proces zahteva.
Kot slabost FCFS razvrščevalnega algoritma omenimo, da algoritem ne podpira prekinjanja, to pomeni, da ko enkrat proces zasede procesor, ga ne izpusti, dokler sam proces tega ne zahteva (konec procesa ali premestitev izvajanja na V/I napravo.
Zamislimo si primer, ko imamo v sistemu en dolg proces, ki se izvaja v procesorju in več kratkih procesov, ki so v V/I vrsti. Ko prihaja do prehoda procesov se lahko zgodi naslednja situacija. Dolg proces se izvaja v procesorju, ostali procesi, ki so končali izvajanje na V/I napravi se postavijo v pripravljeno čakalno vrsto in čakajo, da se dolg proces na procesorju zaključi (!delujemo brez prekinjanja! (non preemptive način)). Ta čas so V/I naprave brez dela. Proces konča delo na procesorju in se postavi v V/I čakalno vrsto in se začne izvajati. Ostali kratki procesi pridejo na vrsto in se izvedejo in postavijo v V/I čakalno vrsto in zopet čakajo na dolg proces, ki se sedaj izvaja na V/I napravi. Sedaj je brez dela procesor, ker vsi procesi čakajo v V/I čakalni vrsti. Temu problemu pravimo **efekt konvoja**., kjer hitri procesi čakajo počasnega in je medtem enkrat procesor neizkoriščen, drugič pa strojna oprema.

5.1.4.2. *"Najkrajše opravilo najprej" algoritem razvrščanja (Shortest-Job-First – SJF)*

Algoritem izbere proces, ki ima trenutno v pripravi najkrajši računski del – cikel na procesorju. Če imata dva procesa enak računski del se uporabi algoritem FCFS.
Kot primer si vzemimo nabor procesov, ki pridejo ob času 0, s ciklom na procesorju podan v ms.

Proces	Cikel
P_1	6
P_2	8
P_3	7
P_4	3

Če procesi pridejo v zaporedju P_1, P_2, P_3, P_4, jih bo algoritem SJF dodeljeval procesorju kot kaže naslednja slika.



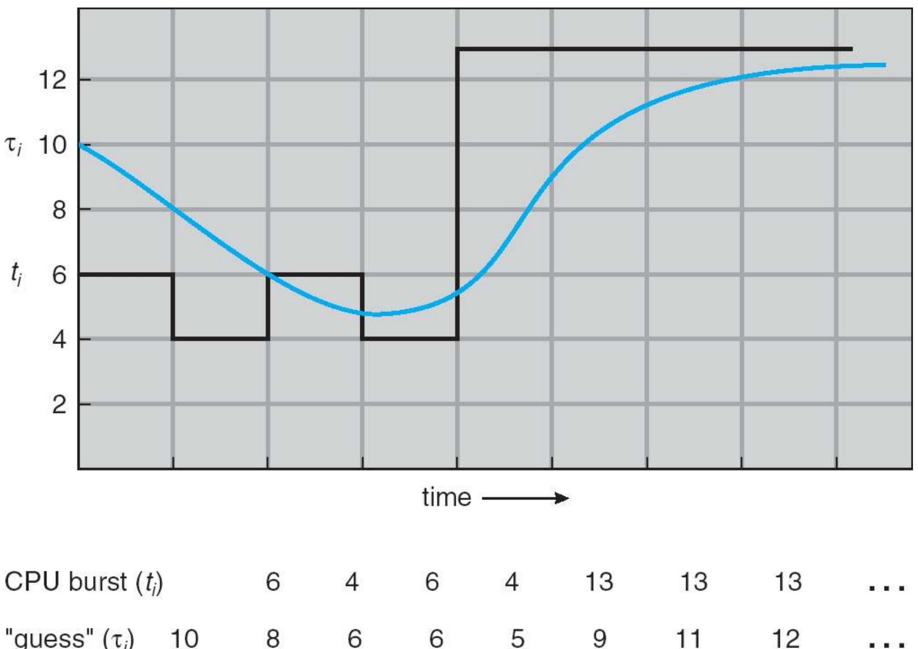
Povprečen čakalni čas je v tem primeru (0 ms+3 ms+9 ms+16 ms)/4 = **7 ms**. Če bi uporabili FCFS algoritem, bi bil ta čas **10,25 ms**.

Problem SJF algoritma je poznavanje dolžine cikla na procesorju, ki ga bo porabil naslednji proces. Pri dolgoročnem razvrščevanju pri paketnih sistemih, lahko dolžino naslednjega opravila določi kar uporabnik, ko preda sistemu opravilo. Pri kratkoročnem razvrščanju je SJF algoeitem zelo težko aplicirati, saj ni mogoče ugotoviti dolžine cikla naslednjega procesa.
Natančno ne moremo vedeti dolžine naslednjega cikla, lahko jo pa napovemo z **eksponentnim povprečjem dolžin** prejšnjih ciklov na procesorju. Eksponentno povprečje napišemo s formulo:

$$\tau_{n+1}=\alpha t_n+(1-\alpha)\tau_n$$

Kjer je τ_{n+1} napovedana dolžina naslednjega cikla, τ_n dolžina trenutnega ciklla, α relativna teža prejšnjih in prihodnjih dolžin ciklov.
Izraz αt_n vsebuje najnovejše informacije o dolžini cikla za ta proces, izraz $(1-\alpha)\tau_n$ pa vsebuje zgodovino naših napovedi.

Poglejmo še kako natančno določimo dolžino naslednjega cikla preko prejšnje formule. Pri tem imamo parametre α= 0,5 in τ_0=10 ms.



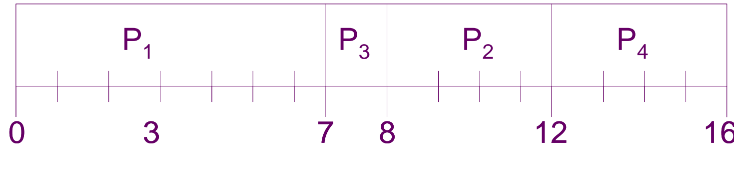
SJF

algoritem lahko deluje s prekinjanjem ali brez prekinjanja. Potreba po izbiri naraste, ko pride nov proces v pripravljeno vrsto, medtem, ko se prejšnji proces še izvaja. Nov proces ima lahko krajši cikel, kot je še preostalo časa tekočemu procesu do konca izvajanja. **Algoritem SJF s prekinjanjem** bi prekinil tekoči proces, **algoritem SJF brez prekinjanja** pa bi pustil tekoči proces, da se izvede do konca. Algoritmu SJF s prekinjanjem pravimo tudi **Prednost nalogi s krajšim ostankom časa** (Shortest Remaining Time First – SRTF).

Primer algoritma SJF brez prekinjanja.

Proces	Prihod	Cikel
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

V tem primeru ni prekinjanja tekočega procesa in povprečni čakalni čas izračunamo:

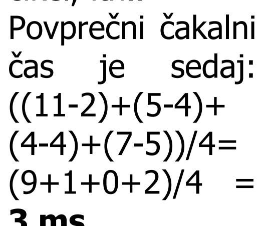


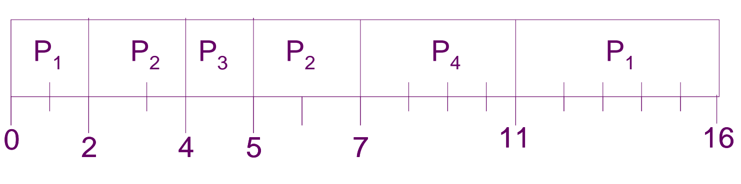
(0+(8-2)+(7-4)+(12-5))/4=(0+6+3+7)/4 = **4 ms**

Primer algoritma SJF s prekinjanjem ali SRTF algoritem.

Proces	Prihod	Cikel
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

Po algoritmu s prekinjanjem pridemo do naslednje slike. Najprej se začne osamljen proces P_1 izvajati, vendar čez 2 ms pride P_2, ki ima cikel 4 ms, P_1 pa do konca ostaja še 5 ms. Torej po naravi algoritma SRTF mora priti do prekinitve P_1 in izvajanja P_2, ker ima krajši cikel, itn..


Povprečni čakalni čas je sedaj: ((11-2)+(5-4)+(4-4)+(7-5))/4= (9+1+0+2)/4 = **3 ms**.



5.1.4.3. *Prioritetno razvrščanje*

SJF algoritem je poseben primer prioritetnega razvrščanja, kjer se prioriteta (celoštevilčna vrednost) določa posameznemu procesu. Prioriteta je obratnosorazmerna ciklu procesa na procesorju. Večji kot je cikel manjša je prioriteta procesa.
Prioritete so definirane interno ali eksterno. Interna prioriteta je odvisna od časovnih omejitev, zahtev pomnilnika, števila odprtih datotek, razmerja povprečnega V/I cikla in cikla na procesorju. Eksterne prioritete določajo kriteriji, ki so zunanji operacijskemu sistemu, kot so pomembnost procesa in nenazadnje tudi politični vplivi. Omenimo še, da lahko uporabnik direktno spreminja prioriteto procesov(npr. operacijski sistem Linux).
Problem prioritetnih algoritmov (SJF, SRTF) nastane, kadar je veliko procesov z majhnim procesorskim ciklom in algoritem pusti procese z daljšim procesorskim ciklom čakati in tako ne morejo priti na vrsto, da bi se začeli izvajati. Temu problemu pravimo **nedoločeno blokiranje ali stradanje procesa**. Na zelo obremenjenih sistemih bi to pomenilo, da bi se proces z nižjo prioriteto izvršil v Nedeljo ob 14h, ko na sistemu ni veliko dela.
Problem rešuje tehnika **staranje** (aging), ki poskrbi, da procesi, ki čakajo - stradajo nek določen čas pridobijo višjo prioriteto. Kot primer imamo razpon prioritet od 0 do 127. Tehnika bi vsakih 15 min procesu, ki strada povišala prioriteto za 1. Tak proces se bo če ne prej izvedel čez 15 min * 127 = 31 h in 45 min.

5.1.4.4. *Round Robin (RR) razvrščanje*

RR razvrščevalni algoritem je narejen posebej za večopravilne sisteme. Podoben je FCFS algoritmu, le da je RR algoritmu dodano prekinjanje za preklop med procesi.
Definirajmo majhno enoto časa imenovano tudi **časovna rezina**. Časovna rezina ima običajno vrednosti od 10 do 100 ms. Pripravljena vrsta je pri tem algoritmu tretirana kot vrsta v obliki kroga. Razvrščevalnik procesorja gre okoli pripravljene vrste, kjer dodeli procesorju del vsakega procesa za čas 1 časovne rezine.

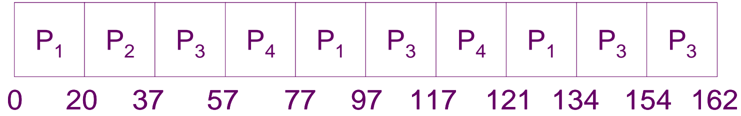
Implementacija RR razvrščanja se začne pri pripravljeni vrsti, ki deluje po principu FIFO. Novi procesi se postavijo na konec pripravljene vrste. Razvrščevalnik procesorja pobere prvi proces v pripravljeni vrsti, postavi časovnik, ki sproži prekinitev čez 1 časovno rezino (od 10 do 100 ms) in proces dodeli procesorju (dodeljevalnik - "dospatcher").
Ena od dveh stvari se lahko zgodi.

- Cikel procesa, ki ga bi proces porabil, ko bi se izvajal je krajši od ene časovne rezine. V tem primeru bi proces samovoljno zapustil procesor in razvrščevalnik bi nadaljeval na naslednjem procesu v pripravljeni vrsti.
- Cikel procesa, ki ga bi proces porabil, ki bi se izvajal je daljši od ene časovne rezine. V tem primeru bi časovnik po času 1 časovne rezine sprožil prekinitev operacijskemu sisitemu. Sprožil bi se menjevalnik okolja (context switch), proces bi bil postavljen na konec pripravljene vrste. Razvrščevalnik bi potem izbral naslednji proces iz pripravljene vrste.

Povprečni čakalni čas pri RR algoritmu je ponavadi dolg.
Zamislimo si naslednje zaporedje procesov, ki prispejo ob času 0, z dolžino cikla na procesorju podanega v ms.

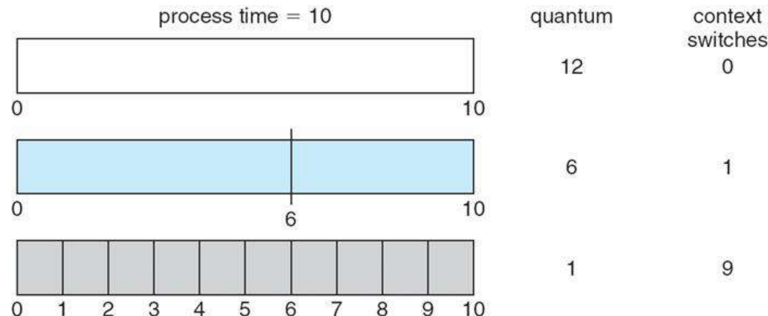
Proces	**Cikel**
P_1	53
P_2	17
P_3	68
P_4	24

Predpostavimo, da RR algoritem uporablja: 1 časovna rezina = 20 ms. Proces P1 naj bi se na procesorju izvajal 53 ms in ker je daljši od ene časovne rezine ga razvrščevalnik prekine po 20 ms (1 časovni rezini). Procesorju je dodeljen proces naslednji proces P2, ki pa je krajši od ene časovne rezine in zato samovoljno konča delo na procesorju po 17 ms ter sprosti pripravljeno vrsto. Procesorju je dodeljen naslednji proces P3 in ko preteče 1 časovna rezina pride zopet do prekinitve, itd. Ko pridemo do zadnjega razvrščevalnik ponovno dodeli proces P1 in ta se zopet prekine čez 20 ms. Rezultirajoče RR razvrščanje je prikazano na sliki.

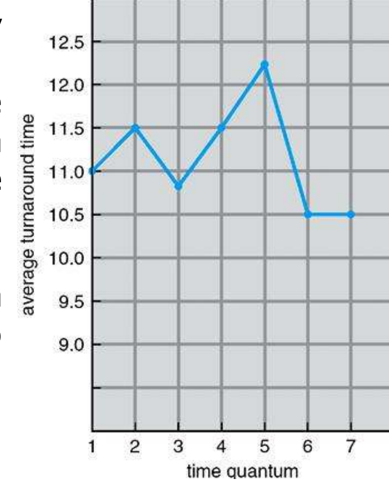


Povprečen čakalni čas je v tem primeru ((77+44)+(20)+(37+60+37)+(57+60))/4 = (121+20+148+117)/4= **101,5 ms**.
V RR algoritmu ni noben proces dodeljen procesorju za več kot 1 časovno rezino. RR algoritem podpira prekinjanje.
Če je v pripravljeni vrsti n procesov in imamo q dolgo časovno rezino potem vsak proces zasede 1/n časa procesorja po kosih q. Vsak proces ne more čakati več kot (n-1)*q da se ponovno dodeli procesorju. Na primer, če imamo 5 procesov s časovno rezino 20 ms, potem noben proces ne bo čakal dlje kot 80 ms da se ga ponovno dodeli procesorju.
Učinek RR algoritma je zelo odvisna od velikosti časovne rezine. Slika kaže kako velikost časovne rezine vpliva na število menjav okolja (context switch). Menjava okolja vpliva na

učinkovitost RR algoritma. V prvem primeru se ne zgodi nobena menjava okolja, v drugem se zgodi 1 menjava okolja v tretjem pa kar 9 menjav okolja, ki seveda upočasnijo učinkovitost algoritma.



Zato hočemo, da bo časovna rezina večja v primerjavi s časom, ki ga porabi menjava okolja. Če je čas menjave okolja 10 % časa ene časovne rezine pomeni, da bo 10 % časa procesorja porabljenega za izvajanje menjave okolja.
Tudi čas obdelave (turnaround time) je precej odvisen od časa časovne rezine. Kot vidimo na sliki se povprečni čas obdelave ne bistveno izboljša, če se čas časovne rezine daljša.



process	time
P_1	6
P_2	3
P_3	1
P_4	7

5.1.4.5. Več-nivojska vrsta razvrščanja

Algoritem je primeren za situacije, kjer se procese da razdeliti v določene skupine. Na primer osnovna delitev bi lahko bila delitev na **procese v ozadju** (paketni procesi – tiskanje iz urejevalnika besedil) in **procese v ospredju** (interaktivni procesi – delo z V/I napravami). Ponavadi imajo procesi v ospredju višjo prioriteto kot procesi v ozadju.
Ta dva tipa procesov imata različne časovne odzive in imata različne zahteve po razvrščevalnih algoritmih.
V večnivojski vrsti so procesi najprej umeščeni v vrste, kateri procesi po lastnostih najbolj pripadajo. Vsaka vrsta ima svoj razvrščevalni algoritem in višja vrsta ima vedno prednost pri izvajanju pred nižjimi vrstami.
Če se na primer izvaja proces v paketni vrsti (batch) in med izvajanjem procesa v paketni vrsti prode proces, ki spada v višjo vrsto bo proces v paketni vrsti prekinjen in izvajati se bo začel proces v višji vrsti.
Druga možnost je, da se doda vsaki vrsti določen čas izvajanja (npr. 80% časa CPE višjim vrstam (RR algoritem razvrščanja) in 20% časa nižjim vrstam (FCFS algoritem razvrščanja)).

5.1.4.6. Primer Windows XP

Niti so tukaj razvrščene po prioritetnem principu in algoritmih, ki temeljijo na prekinitvah. XP razvrščevalni algoritem zagotavlja, da bo nit z najvišjo prioriteto vedno deležna procesorja. Del jedra operacijskega sistema Windows XP, ki upravlja razvrščanje se imenuje dodeljevalnik (dispatcher).
Nit, ki je izbrana bo dodeljena procesorju, dokler:

- ne dobi prekinitve od niti z višjo prioriteto,
- se ne sama ne zaključi,
- ne preteče čas časovne rezine ali
- ne izvede blokirnega sistemskega klica (V/I operacija)

Dodeljevalnik uporablja 32 nivojev prioritetnih shem, ki definirajo katera nit se bo izvedla. Prioritete so razdeljene v dva razreda:

- spremenljivi razred, ki vsebuje niti s prioriteto od 1 do 15 in
- realno-časovni razred, ki vsebuje niti s prioriteto od 16 do 31.

Tule so tudi niti, ki tečejo s prioriteto 0, ki se uporablja za upravljanje pomnilnika.
Dodeljevalnik uporablja vrsto za vsako razvrščevalno prioriteto in potuje med prioritetnimi vrstami od višje proti nižji, dokler ne najde niti, ki je pripravljena na izvajanje. Če dodeljevalnik ne najde nobene niti pripravljene na izvajanje v nobeni prioritetni vrsti dodeli procesor tako imenovani **brezdelni niti** (idle threat).

	real-time	high	above normal	normal	below normal	idle priority
time-critical	31	15	15	15	15	15
highest	26	15	12	10	8	6
above normal	25	14	11	9	7	5
normal	24	13	10	8	6	4
below normal	23	12	9	7	5	3
lowest	22	11	8	6	4	2
idle	16	1	1	1	1	1

5.1.4.7. Primer Linux

Od jedra 2.5 naprej Linux razvrščevalnik deluje na osnovi prekinitvenih in prioritetnih algoritmov. Linux deli dva prioritetna razreda: realno-časovni razred (0-99) in prijazni razred (100-140). Linux dodeli procesom z višjo prioriteto več časa procesorja kot procesom z nižjo prioriteto.

5.2. Vprašanja za ponavljanje

1. Kaj je razvrščevalnik na CPE?

2. Kaj je naloga dodeljevalnika?

3. Zamislite si, da so v sistem istočasno prišli procesi:

Proces	Čas cikla	Prioriteta
P1	10	3
P2	1	1
P3	2	3
P4	1	4
P5	5	2

a.) Nariši grafe, ki ilustrirajo izvajanje procesov z uporabo FCFS, SJF, prioritetno brez prekinjanja in RR (kvant časa=1 ms).

b.) Kakšen je čas obdelave za vsak proces v vsakem algoritmu?

c.) Kakšni so čakalni in povprečni čakalni časi za vsak algoritem?

d.) Kateri algoritem rezultira v najmanjšem povprečnem času vseh procesov?

4. Opiši razvrščanje z večnivojskimi vrstami.

5.3. Sinhronizacija procesov

Sodelujoč proces je tisti proces, ki lahko vpliva na druge procese in drugi procesi lahko vplivajo nanj.
Sodelujoči procesi si lahko delijo:

- logični naslovni prostor (kodo programa in podatke) ali
- samo podatke preko datotek

Problem, ki se lahko pojavi pri sodelujočih procesih je **nekonsistentnost podatkov**. Zato je cilj poglavja spoznati probleme, pri katerih lahko pride do nekonsistentnosti skupnih podatkov in mehanizme, ki bi probleme nekonsistentnost skupnih podatkov odpravljali.

5.3.1. Ozadje problema - nekonsistentnost skupnih podatkov

V poglavju o procesih smo govorili o sistemu v katerem je več sodelujočih, zaporednih procesov, kateri se izvajajo asinhrono.
Rešitev, ki smo jo predstavili v poglavju o sodelujočih procesih omogoča največ `VELIKOST_BUFFERJA-1` poln "buffer". To pomanjkljivost, bi radi odpravili.
Uvedli bi lahko spremenljivko `stevec`, ki bi se povečal za 1, ko bi v "buffer" vpisali novo informacijo, in zmanjšal za 1, ko bi informacijo iz njega prebrali.

Kodo proizvajalca bi spremenili takole:

```
while (1) {
      /*proizvedi informacijo v proizvedi_naslednjega */
      while (stevec == VELIKOST_BAFERJA)
      ; /*ne naredi ničesar*/
      bafer[noter]= proizvedi_naslednjega;
      noter=(noter+1)%VELIKOST_BAFERJA;
      stevec++;
}
```

Kodo porabnika bi spremenili takole:

```
while (1) {
      while (stevec==0)
      ; /*ne naredi ničesar*/
      porabi_naslednjega = bafer[ven];
      ven=(ven+1)%VELIKOST_BAFERJA;
      stevec--;
      /* porabi informacijo v porabi_naslednjega */
}
```

Tudi, če sta obe kodi pravilno ločeno lahko pride do nekonsistentnosti, ko delujeta sočasno!
Kot primer si zamislimo vrednost spremenljivke `stevec`=5 in da procesa proizvajalec in porabnik izvajata istočasno stavek `stevec++` in `stevec--`. Vrednost spremenljivke `stevec` je lahko po izvršenem stavku 4, 5 ali 6. Pravilna vrednost bi bila 5, če bi se koda proizvajalca in porabnika izvršila ločeno.

Pokažimo primer nepravilne vrednosti v spremenljivki **stevec**.
Stavek **stevec++** zgleda na običajnem računalniku takole:

```
register1=stevec
register1=register1+1
stevec=register1
```

Ker je **register1** register v procesorju, kjer se operacija izvede. Stavek stevec-- zgleda na običajnem računalniku takole:

```
register2=stevec
register2=register2+1
stevec=register2
```

Kjer je **register2** drugi register v procesorju, kjer se operacija izvede.

Istočasna izvedba stavkov bi lahko pripeljala do tega:

T_0:	proizvajalec	izvede	$register_1$=stevec	{$register_1$=5}
T_1:	proizvajalec	izvede	$register_1$=$register_1$+1	{$register_1$=6}
T_2:	porabnik	izvede	$register_2$=stevec	{$register_2$=5}
T_3:	porabnik	izvede	$register_2$=$register_2$-1	{$register_2$=4}
T_4:	proizvajalec	izvede	stevec=$register_1$	{$register_1$=6}
T_5:	porabnik	izvede	stevec=$register_2$	{$register_1$=4}

Tudi, če bi se zadnja dva koraka izvedla v obratnem vrstnem redu, bi prišli do napačnega stanja bufferja.
Števec kaže, da je register poln do 4. Realno pa imamo polnega do 5. Do tega smo prišli, ker smo dovolili obema procesoma, da sta uporabljala istočasno spremenljivko **stevec**.
Taka situacija, ko več procesov sočasno spreminja iste podatke in je rezultat izvajanja odvisen od določenega vrstnega reda, po katerem procesi dostopajo do podatkov, je neke **vrste tekma** (race condition) in ni zaželena.
Da preprečimo take tekme moramo preprečiti, da se istočasno uporablja skupno spremenljivko. Take situacije se dogajajo pogosto v operacijskih sistemih.

5.3.2. Problem kritične sekcije

Dan je sistem z n procesi (P_0, P_1, ... ,P_{n-1}). Vsak od procesov ima del kode, imenovan **kritična sekcija** (critical section), v kateri lahko proces spreminja skupne spremenljivke, posodablja tabele, piše v datoteko, ... Pomembna stvar takega sistema je, ko en proces izvaja svojo kritično sekcijo, ne sme noben drug proces izvajati svoje. Torej je izvajanje kritičnih sekcij procesov časovno vzajemno izključujoče.
Problem kritičnih sekcij je izdelati protokol, ki bi ga procesi uporabljali z namenom, da bi med seboj sinhrono delovali.
Vsak proces mora zaprositi za dovoljenje za vstop v svojo kritično sekcijo in je sestavljen iz:

- kode, ki implementira prošnjo za vstop v kritično sekcijo in se imenuje **vstopna sekcija** (entry section).
- kode, ki se izvede, ko proces izstopi iz kritične sekcije in se imenuje **izstopna sekcija**

(exit section).

- kode, ki izvede preostali del programa in se imenuje **preostala sekcija** (reminder section).

Rešitev problema kritičnih sekcij mora zadoščati naslednjim zahtevam:

- **Vzajemno izključevanje** (**mut**ual **ex**clusions) **–** če se proces P_i izvaja v svoji kritični sekciji, potem se noben drug proces ne sme izvajati v svoji kritični sekciji
- **Napredovanje** (progress) **–** če ni procesa v svoji kritični sekciji in obstaja nekaj procesov, ki želijo vstopiti v svojo kritično sekcijo, potem ti procesi sodelujejo pri odločanju, kateri bo vstopil v svojo kritično sekcijo.
- **Omejeno čakanje** (bounded waiting) **-** proces mora v končnem času vstopiti v svojo kritično sekcijo od trenutka, ko je dal prošnjo za vstop v svojo kritično sekcijo do trenutka, ko dobi dovoljenje za vstop v svojo kritično sekcijo.

Predpostavimo, da se osnovni ukazi strojnega jezika (load, store, test, add, ...) izvedejo **atomarno** (nedeljivo) – morajo se izvesti do konca, preden se upošteva želena prekinitev. Poleg tega pa mora biti rešitev neodvisna od vrednosti n (se pravi števila procesov) in tehnoloških značilnosti računanja.

Rešitev problema kritičnih sekcij, kjer so zadoščeni trije pogoji se lotimo na sistemu z **dvema procesoma**.

5.3.2.1. Algoritem 1

Prvi poizkus je v tem, da si procesa P_i in P_j delita skupno celoštevilčno spremenljivko `na_vrsti`. Spremenljivka `na_vrsti` je inicializirana z 0 (ali 1). Če velja `na_vrsti==i`, potem velja, da je na vrsti proces P_i in P_i lahko izvede svojo kritično sekcijo.
Algoritem ugodi zahtevi vzajemnega izključevanja, ne daje pa dovolj informacij, saj če velja, če je spremenljivka `na_vrsti=0` in P_1 pripravljen za vstop v svojo kritično sekcijo, P_1 ne more vstopiti v kritično sekcijo, čeprav je P_0 že v izhodno sekciji ali celo v preostali sekciji.

```
do {
      vstopna sekcija
            kritična sekcija
      izstopna sekcija
            preostala sekcija
} while (1);
```

5.3.2.2. Algoritem 2

Problem algoritma 1 je bil, da ne daje dovolj informacij v kakšnem stanju izvajanja je proces, ki ima dovoljenje za izvajanje svoje kritične sekcije, zapomni si samo, kateri proces lahko izvaja svojo kritično sekcijo.

Če zamenjamo spremenljivko **`na_vrsti`** z poljem

`boolean zastavica[2];` /*polje z dvema elementoma, ki sta lahko true ali false */

dobimo informacijo v kakšni fazi izvajanja je določen proces.

Elementa polja imata na začetku vrednost `false`. Če `zastavica[i]` ima vrednost true pomeni, da je P_i pripravljen za vstop v svojo kritično sekcijo.

Proces P_i	Proces P_j

do {
    zastavica[i]=true;
    while (zastavica[j]);
        kritična sekcija
    zastavica[i]=false;
        preostala sekcija
} **while (1);**
	do {
	    zastavica[j]=true;
	    while (zastavica[i]);
	        kritična sekcija
	    zastavica[j]=false;
	        preostala sekcija
	} **while (1);**

zastavica[i]	false/true
zastavica[j]	false/true

V tem primeru P_i postavi ***zastavica[i]=true;***, ki signalizira, da je pripravljen za vstop v svojo kritično sekcijo.
Nato P_i preveri ali je tudi P_j pripravljen za izvajanje svoje kritične sekcije (***while (zastavica[j]);***), če je, P_i počaka, dokler P_j ne signalizira, da je končal izvajanje svoje kritične sekcije z stavkom (***zastavica[j]=false;***), če Pj ni pripravljen, Pi vstopi v svojo kritično sekcijo in ob zaključku izvajanja kritične sekcije takoj signalizira, da je končal z (***zastavica[i]=false;***).
Problem je v napredovanju procesov.
Zamislimo si naslednje sekvence izvajanja stavkov:

T_0: P_0 postavi ***zastavica[0]=true***
T_1: P_1 postavi ***zastavica[1]=true***

Vsak proces neskončno preverja, če je drug proces signaliziral, da je končal izvajanje kritične sekcije – **smrtni objem** (deadlock).

5.3.2.3. Algoritem 3

Z združitvijo dobrih lastnosti obeh algoritmov pridemo do prave rešitve.
Proces si izmenjuje informacije v dveh spremenljivkah:

```
boolean zastavica[2];
int na_vrsti;
```

Na začetku je
zastavica[0]=false
zastavica[1]=false;
na_vrsti=0;

Proces P_0	Proces P_1

`do {`
`  zastavica[0]=true;`
`  na_vrsti=1;`
`  while(zastavica[1] && na_vrsti==1);`
`        kritična sekcija`
`  zastavica[0]=false;`
`        preostala sekcija`
`} while (1);`
	`do {`
	`  zastavica[1]=true;`
	`  na_vrsti=0;`
	`  while(zastavica[0] && na_vrsti==0);`
	`        kritična sekcija`
	`  zastavica[1]=false;`
	`        preostala sekcija`
	`} while (1);`

Omenili smo rešitve na dveh procesih. Poznamo tudi in rešitve na n procesih – "**bakery algoritm"**. Vse rešitve so programske narave.

5.3.3. Sinhronizacija preko strojne opreme

Zgornji algoritmi so bili programski. Programsko reševanje problema kritičnih sekcij pa je časovno prezahtevno. Poleg tega pa strojne rešitve ponavadi naredijo programske naloge lažje in izboljšajo učinkovitost. Zato ponekod obstajajo posebni ukazi, s katerimi lahko dosežemo učinkovitejše rešitve problema kritičnih sekcij.

5.3.3.1. Onemogočanje prekinitev (interrupt disabling)

Če onemogočimo prekinitve, ki se pojavljajo med spreminjanjem skupne spremenljivke, zagotovimo nemoteno izvajanje kritične sekcije, ter pravilni vrstni red izvajanja zaporednih stavkov.
Struktura procesa Pi:

```
onemogoči prekinitev;
      kritična sekcija
omogoči prekinitev;
```

Onemogočanje prekinitev nas lahko hitro pripelje do veliko novih težav.

- Delo v realnem času je tako onemogočeno, ko se med izvajanjem kritične sekcije zgodi potreba po spremembi okolja.
- V primeru, da se v kritični sekciji izvaja dolga zanka, pride do **ciklanja** ali **prisvojitve procesorja** in seveda izgube časa.

5.3.3.2. Strojni ukaz TestAndSet

Običajna strojna oprema omogoča izvajanje posebnega strojnega ukaza, ki nam dovoli preverjanje in spreminjanje vsebine besede na atomaren način.
`TestAndSet` ukaz lahko definiramo kot kaže spodnja slika.

```
function TestAndSet (var x:boolean):boolean;
      begin
            TestAndSet:=x;
            x:=true;
      end
```

Instrukcija `TestAndSet` se mora izvesti **atomarno**–nedeljivo!! Atomarnost lahko zagotovimo z onemogočanjem prekinitev.

Vzajemno izključevanje lahko implementiramo z deklaracijo spremenljivke `nekdo_v_ks`. (nekdo v kritični sekciji?)

```
var nekdo_v_ks:boolean;
nekdo_v_ks:=false;

do {
  while (TestAndSet(nekdo_v_ks));
      kritična sekcija
  nekdo_v_ks=false;
      preostala sekcija
} while (1);
```

Ta rešitev zajema vzajemno izključevanje, lahko se zgodi, da P_i čaka, da P_j izstopi iz svoje kritične sekcije, medtem pa pride proces P_k, ki tudi čaka na proces P_j. Ko P_j konča, vstopi v svojo kritično sekcijo P_k in P_i še vedno čaka. P_i lahko čaka v neskončnost, če vedno znova prihajajo procesi in ga prehitijo. Iz tega sledi, da lahko nek proces čaka na izvajanje v procesorju zelo dolgo – staranje (Preprečevanje staranja posega v operacijski sistem).

Obstaja tudi strojna instrukcija `swap`, ki je alternativa instrukciji `TestAndSet`, ki deluje na spremembi vsebine dveh besed.

5.3.4. Semaforji

Rešitev problemu kritične sekcije s strojno opremo ni enostavno izvesti pri bolj kompleksnih sistemih oz. problemih, zato uporabimo sinhronizacijsko orodje imenovano **semaforji**.
Semafor je celoštevilčna spremenljivka, katero lahko spreminjano samo preko dveh **atomarnih operacij**: `wait(S)` in `signal(S)`.

Definicija `wait(S)`	Definicija `signal(S)`

`wait(S) {`
`      while (S<=0)`
`      ;//no-op`
`      S--;`
`}`
	`signal(S) {`
	`      while (S)`
	`      ;//no-op`
	`      S++;`
	`}`

Ko nek proces spremeni vrednost semaforja, ne sme noben drug proces spreminjati te vrednosti. Enako velja za atomarne operacije `wait(S)` in `signal(S)`, še posebej ko se izvaja operacija `S--` ali `S++`.

5.3.4.1. Uporaba semaforjev

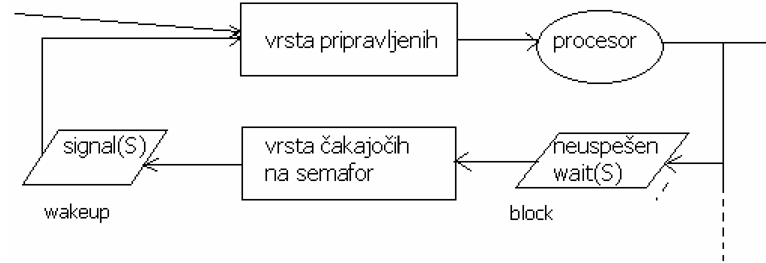
Semaforje lahko uporabimo pri reševanju problema kritičnih sekcij z več – n procesi. Ideja je v temu, da si procesi delijo semafor **mutex** (**mut**ual **ex**clusions), ki ima začetno vrednost 1. Vsak proces P_i je organiziran kot kaže slika.

```
do {
  wait(mutex);
      kritična sekcija
  signal(mutex);
      preostala sekcija
} while (1);
```

5.3.4.2. Implementacija semaforjev

Glavna pomanjkljivost v primeru reševanja kritičnih sekcij z algoritmi in prikazanem primeru s semaforjem `mutex` je to, da zahtevajo **nekoristno čakanje** (busy waiting). Ko se nek proces nahaja v svoji kritični sekciji in ostali procesi čakajo na vstop v svojo kritično sekcijo, morajo čakajoči procesi izvajati zanko čakanja (loop) v svoji vstopni sekciji. Neprestano čakanje v svoji vstopni sekciji je velik problem pri realnem večopravilnem sistemu, kjer si en procesor deli več procesov.
Nekoristno čakanje zapravlja cikle (čas) procesorja, medtem, ko bi lahko ta čas s pridom izkoristili čakajoči procesi. Taka uporaba semaforja se imenuje tudi krožni semafor (spinlock).
Da odpravimo nekoristno čakanje spremenimo definicijo `wait(S)` in `signal(S)` operacije semaforja.
Ko proces izvaja `wait(S)` operacijo in ugotovi, da vrednost semaforja ni pozitivna mora čakati. Raje kot nekoristno čakanje (obremenjevanje procesorja z `loop` operacijo), proces blokira samega sebe.
Blok operacija (block operation) prestavi proces iz pripravljene vrste v čakalno vrsto – (status procesa se spremeni v čakajoč!). Nato se kontrola prenese na razvrščevalnik procesorja, ki izbere naslednji proces v pripravljeni vrsti in ga dodeli procesorju – ni nekoristnega čakanja!! Proces , ki je blokiran v čakalni vrsti in čaka na semafor S mora biti ponovno zagnan, ko drug proces izvede `signal(S)` operacijo. Proces se ponovno zažene in se prestavi iz čakalne vrste v pripravljeno vrsto z **operacijo zbudi** (wakeup operation).
Po tej definiciji lahko definiramo semafor kot semafor, ki vsebuje celoštevilčno vrednost in seznam procesov.

```
typedef struct {
      int vrednost;
      struct process *L;
} semaphore;
```



Ko proces čaka na semafor, je proces zapisan na seznamu procesov. Operacija **signal(S)** prenese en proces iz čakalne vrste in ta proces zbudi.

wait semafor definiramo z:

```
void wait(semaphore S){
      S.value--;
      if (S.value<0) {
            dodaj ta proces v S.L (seznam procesov);
            block();
      }
}
```

signal semafor definiramo z :

```
void signal(semaphore S){
      S.value++;
      if (S.value<=0) {
            odstrani proces P iz S.L (seznama procesov);
            wakeup(P);
      }
}
```

Operacija **block()** začasno ustavi proces, ki je zaprosil za izvajanje kritične sekcije in ga postavi v čakajočo vrsto. Operacija **wakeup(P)** nadaljuje prekinjeno izvajanje začasno ustaljenega procesa in ga postavi v pripravljeno vrsto.
Ti dve operaciji izvede operacijski sistem preko **sistemskega klica**.

Pri tej implementaciji semaforja je vrednost semaforja lahko negativna, kar pri prejšnjem primeru ne mora biti. Velikost negativne številke predstavlja število čakajočih procesov na semafor. Seznam čakajočih procesov na semafor lahko enostavno izvedemo z povezanim poljem v PCB. Vsak semafor vsebuje celoštevilsko vrednost in kazalec na seznam procesov v čakalni vrsti. Procesi se praznijo iz čakalne vrste po načelu FIFO (First In First Out).

Kritični pogled na semaforje je ta, da izvaja operaciji **wait(S)** in **signal(S)** atomarno -nedeljivo. Zagotoviti moramo, da ne pride do tega, da bi dva procesa izvedla **wait(S)** in **signal(S)** operacijo na istem semaforju istočasno. Ta problem je problem kritičnih sekcij in ga lahko rešimo na dva načina.

- v enoprocesorskem sistemu lahko preko **prekinitve** zagotovimo, da se izvaja samo **wait(S)** ali samo **signal(S)** operacija. Tak pristop deluje samo v enoprocesorskem sistemu, ker ko enkrat pride do prekinitve, se instrukcije drugih procesov ne morejo izvajati. Izvaja se lahko samo trenutni proces, dokler prekinitve niso ponovno vzpostavljene in razvrščevalnik zopet pridobi kontrolo.
- v večprocesorskem sistemu zadrževanje z prekinitvami ne deluje. Ukazi drugih procesov , ki tečejo na drugih procesorjih lahko vskočijo v nekem zaporedju. Če strojna oprema ne dopušča kakšne posebne instrukcije si moramo pomagati z programskimi rešitvami - algoritmi, ki smo jih prej omenili.

5.3.5. Smrtni objemi

Implementacija semaforjev in čakalne vrste lahko pripelje do situacije, kjer dva ali več procesov nedoločeno čakajo na dogodek, ki ga lahko sproži eden od čakajočih procesov. Dogodek bi tule bil operacija signal(), ki da signal naslednjemu procesu, da lahko vstopi v svojo kritično sekcijo. Ko pridemo do te situacije pravimo da so procesi v **smrtnem objemu** (deadlocked).
Primer smrtnega objema kaže naslednji zgled:

P_0	P_1

`wait(S);`
`wait(Q);`
.
.
.
`signal(S);`
`signal(Q);`
	`wait(Q);`
	`wait(S);`
	.
	.
	.
	`signal(Q);`
	`signal(S);`

Proces $\mathbf{P_0}$ izvaja wait(S) in nato $\mathbf{P_1}$ izvede wait(Q). Ko P_0 izvede wait(Q) mora čakati dokler $\mathbf{P_1}$ ne izvede signal(Q). Enako velja obratno, ko $\mathbf{P_1}$ izvaja wait(S), mora počakati $\mathbf{P_0}$ da izvede signal(S). Ker eden čaka na drugega sta procesa $\mathbf{P_1}$ in $\mathbf{P_0}$ v smrtnem objemu.

5.3.6. Binarni semaforji

Prej omenjen semafor je **števni semafor**, ker se njegova celoštevilčna spremenljivka giba v neomejenih mejah. Pri binarnem semaforju pa lahko zavzame samo dve vrednosti (0,1). Kako lahko števni semafor predstavimo z binarnim semaforjem?
Naj bo S števni semafor.
Potrebujemo naslednje podatkovne strukture:

```
binarni-semafor S1, S2;
int C;
```

začetne vrednosti: `S1=1, S2=0, C=1;`

wait operacijo definiramo z:

```
wait(S1);
C--;
if (C<0){
      signal(S1);
      wait(S2);
}
signal(S1);
```

signal operacijo definiramo z :

```
wait(S1);
C++;
if (C<=0)
      signal(S2);
      wait(S2);
else
      signal(S1);
```

5.3.7. Klasični problemi sinhronizacije

Nekateri problemi predstavljajo cele družine problemov, zato jih imenujemo »klasični«. Ti problemi se uporabljajo za testiranje skoraj vseh novih sinhronizacijskih rešitev. Za rešitev v sinhronizaciji se uporabljajo semaforji.

5.3.7.1. Problem proizvajalec – porabnik (Bounded-Buffer problem)

Animacija: http://gaia.ecs.csus.edu/~zhangd/oscal/ProducerConsumer/ProducerConsumer.html
Animacija: http://gaia.ecs.csus.edu/~zhangd/oscal/semaphore/semaphore.html

Problem je bil že omenjen na začetku, kot ozadje problema hkratnega izvajanja procesov. Prikazali bomo osnovno strukturo rešitve problema.
Proces P_0 proizvaja podatke, ki jih proces P_1 porablja. Imamo buffer z n mesti.
Imamo tri semaforje:

- semafor **mutex** priskrbi, da ne pride do istočasnega dostopa do istega bufferja in je inicializiran na 1.
- definiramo semafor **prazen**, ki kaže koliko je buffer prazen in je inicializiran na n
- definirajmo semafor **poln**, ki kaže koliko je buffer poln in je inicializiran na 0

kodo za proizvajalca definiramo z:

```
do {
      ...
      proizvedi podatek v naslednji_proizveden
      ...
      wait(prazen);
      wait(mutex);
      ...
      dodaj nextp v buffer
      ...
      signal(mutex);
      signal(poln);
} while (1);
```

kodo za porabnika definiramo z:

```
do {
      wait(poln);
      wait(mutex);
      ...
      prenesi podatek iz bufferja v naslednji_porabljen
      ...
      signal(mutex);
      signal(prazen);
      ...
      porabi podatek v naslednji_porabljen
      ...
} while (1);
```

5.3.7.2. Problem branja-pisanja (Readers-Writers Problem)

Animacija: http://gaia.ecs.csus.edu/~zhangd/oscal/ReaderWriter/ReaderWriter.html

Podatkovni objekt (datoteka ali zapis v bazi podatkov) je lahko v skupni rabi z več istočasnimi ali konkurenčnimi procesi. Nekateri od teh procesov, bi radi samo brali iz podatkovnega objekta, drugi pa tudi posodobili podatek (prebrali in ponovno vpisali v podatkovni objekt). Tako procese ločimo na tiste, ki želijo samo brati – **procesi bralci** (readers) in ostale – **procesi pisalci** (writers).

Če dva procesa pisalca dostopata do podatkovnega objekta istočasno lahko pride do kaosa.

Da bi preprečili kaos, zahtevamo, da ima proces pisec ekskluziven dostop do podatkovnega objekta. Tak sinhronizacijski problem se imenuje problem branja-pisanja. Ko je bil določen kot problem, so na njem testirali vsako sinhronizacijsko rešitev. Ena od teh je rešitev s semaforji.

Poznamo dve vrsti problema:

- **problem branja-pisanja prve vrste**: bralec ne bo čakal na branje, razen če je pisec dobil dovoljenje za pisanje v podatkovni objekt.
- **problem branja-pisanja druge vrste**: zahteva, da ko pisec čaka na dovoljenje za pisanje se mu to omogoči kar se da hitro oz. se nobenemu drugemu bralcu ne dovoli brati, če pisec čaka.

Rešitev problema branja-pisanja prve vrste , proces bralec si deli naslednje podatkovne strukture:

```
semaphore mutex, pisi;
int beristevec;
```

začetne vrednosti: `mutex=1, pisi=1, beristevec=0;`

Semafor pisi je skupen procesu pisec in procesu bralec.

Semafor mutex poskrbi za vzajemno izključevanje ko je spremenljivka `beristevec` posodobljena.

Spremenljivka `beristevec` pove koliko procesov hkrati bere podatkovni objekt.

Semafor pisi deluje kot mutex semafor za procese pisce.

Struktura procesa pisca:

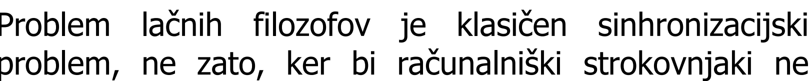
```
wait(mutex);
beristevec++
if (beristevec==1)
      wait(pisi);
signal(mutex);
      ...
      branje podatka
      ...
wait(mutex);
beristevec--;
if (beristevec==0)
      signal(pisi);
signal(mutex);
```

5.3.7.3. Problem lačnih filozofov (Dining-Philosophers Problem)

Animacija: http://www-dse.doc.ic.ac.uk/concurrency/book_applets/Diners.html

Zamislimo si 5 filozofov, ki živijo tako, da mislijo in jejo. Filozofi si delijo okroglo mizo, ki ima 5 stolov. Na sredini mize je skleda riža, 5 krožnikov in 5 kitajskih palčk kot pribor pri jedi.

Ko filozof misli, ne vpliva na druge kolege. V trenutku ko filozof postane lačen želi pobrati dve najbližji kitajski palčki (palčki, ki mejita z desnim in levim kolegom). Filozof lahko naenkrat pobere samo eno kitajsko palčko. Jasno je, da ne more pobrati kitajske palčke, ki jo ima v roki sosednji kolega. Ko lačni filozof dobi obe kitajski palčki jé brez da bi sprostil kitajski palčki. Ko postane sit, sprosti obe kitajski palčki in misli dalje.



Problem lačnih filozofov je klasičen sinhronizacijski problem, ne zato, ker bi računalniški strokovnjaki ne marali filozofov, ampak zato, ker je lep primer problemov, ki se pojavijo pri konkurenčnih istočasnih procesih. Enostavno prikaže potrebo po dodelitvi več sredstev med procesi, kjer ni možno, da pride do stradanja ali smrtnega objema.

Enostavna rešitev leži v temu, da bi vsaki kitajski palčki dodelili semafor. Ko filozof pobere palčko se izvede wait operacija, ko jo sprosti se izvede signal operacija.

Skupni podatki so:

```
semaphore palcka[5];
```

začetne vrednosti semaforja palčk so 1.

Struktura procesa filozofa i:

```
do{
      wait(palcka[i]);
      wait(palcka[(i+1)%5];
      ...
      jé
      ...
      signal(palcka[i];
      signal(palcka[(i+1)%5)
      ...
      misli
      ...
} while(1);
```

Rešitev preprečuje, da bi istočasno dva filozofa pobrala isto palčko, ne preprečuje pa situacije, ko bi vsi filozofi postali lačni in naenkrat in pobrali po eno palčko. Vsak filozof bi čakal, da sosednji filozof odloži palčko. To je seveda situacija, ko bi filozofi zaman čakali na drugo palčko.

Smrtni objem lahko preprečimo če:

- dovolimo največ štirim filozofom, da opravljajo svoje delo istočasno
- dovoliti filozofu, da pobere palčko samo v primeru, da sta obe palčki prosti (to se mora seveda zgoditi v kritični sekciji!!)
- uporabiti asimetrično rešitev – lihi filozof pobere najprej njemu levo palčko in nato desno palčko, dokler sodi filozof pobere najprej njemu desno palčko in nato levo palčko.

5.3.8. Vprašanja za ponavljanje

1. Kaj je kritična sekcija procesa?

2. Kaj pomeni atomarna operacija?

3. Kakšne načine reševanja problema kritične sekcije poznaš?

4. Naštej klasične probleme sinhronizacije in njihove značilnosti.

5. Kaj je smrtni objem in kdaj nastane?

6. Upravljanje shranjevanja

Glavna naloga računalniškega sistema je, da izvaja programe. Programi skupaj s podatki do katerih dostopajo morajo biti v **glavnem pomnilniku** (vsaj delno), če želimo, da se izvajajo.
Da izboljšamo izrabo procesorja in hitrost odziva uporabniku, mora sistem držati več procesov v glavnem pomnilniku. Obstaja veliko shem upravljanja pomnilnika in katero shemo bomo izbrali je odvisno od različnih situacij in faktorjev (še posebej od strojne opreme sistema). Vsaka shema upravljanja pomnilnika ali algoritem zahteva drugačno podporo s strani strojne opreme.
Običajno ima računalniški sistem premalo glavnega pomnilnika, da bi držal vse programe v glavnem pomnilniku, zato mora imeti **sekundarni pomnilnik** v katerega začasno shrani **aktivne** (on-line) programe. Moderni računalniški sistemi uporabljajo **trdi disk** kot sekundarni ali virtualni pomnilnik pomnilnik.
Datotečni sistem priskrbi mehanizem za shranjevanje in branje aktivne vsebine iz glavnega pomnilnika na trdi disk in obratno. Datoteka je skupek povezanih informacij, ki jih je definiral ustvarjalec datoteke. Datoteke so mapirane preko operacijskega sistema na fizičnih enotah. Datoteke so običajno organizirane v mape (direktorije).

6.1. Upravljanje pomnilnika

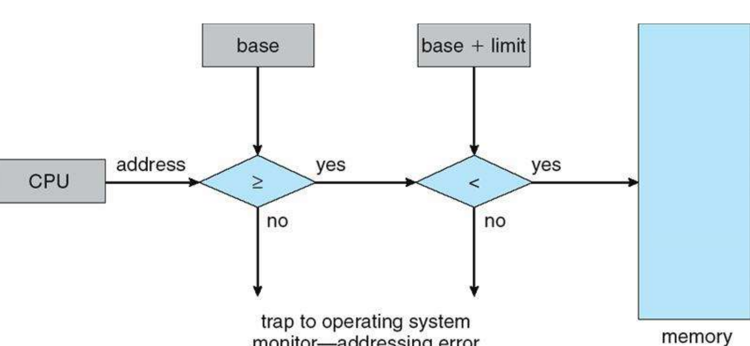
V poglavju razvrščanje na procesorju smo pokazali, kako si procesi delijo procesorski čas. Kot rezultat razvrščanja na procesorju smo izboljšali izrabo procesorskega časa in odziv računalnika njihovim uporabnikom. Za uresničitev teh izboljšav moramo držati več procesov v pomnilniku-**pomnilnik moramo deliti**.
Lahko bi rekli, da je upravljanje pomnilnika naloga operacijskega sistema in strojne opreme, da več procesov razporedi glavnemu pomnilniku.
Spoznali bomo nekaj algoritmov, ki upravljajo s pomnilnikom.
Animacija: http://math.hws.edu/TMCM/java/labs/xComputerLab1.html

6.1.1. Ozadje problema

Pomnilnik je osrednji del pri delovanju modernih operacijskih sistemov. Pomnilnik je sestavljen iz velikega števila **računalniških besed** ali **bytov** (1 byte = 8 bit). Vsaka računalniška beseda domuje na točno določenem **naslovu pomnilnika (address)**.
Tipičen instrukcijski cikel prenese instrukcijo iz pomnilnika na naslovu, ki ga kaže trenutno stanje **programskega števca (PC)**. Instrukcija se prenese v instrukcijski register v procesorju, kjer se instrukcija dekodira in izvede določene poteze (poveča števec, naloži v pomnilnik vrednost, ...). Take instrukcije povzročijo dodatno nalaganje operandov iz oz. shranjevanje rezultatov v določene pomnilniške lokacije.
Pomnilnik vidi tako samo tok naslovov pomnilnika, ne ve kako se je naslov generiral (instrukcijskim števcem, indexiranjem, ...) ali kaj vsebuje naslovljena lokacija (instrukcijo, naslov ali vrednost). Ne zanima nas, kako je program generiral naslov, ampak zaporedje naslovov, ki jih je tekoči program generiral.

6.1.2. Osnovna strojna oprema

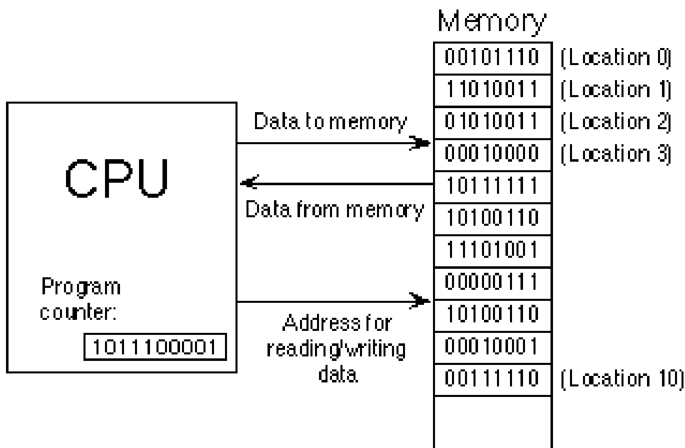
Glavni pomnilnik in registri v procesorju so **edini** shranjevalni prostor kamor lahko procesor **dostopa direktno**. Obstajajo strojne instrukcije, ki vsebujejo vrednosti in pomnilniške naslove kot argumente (npr. instrukcija load 2 – naloži iz naslova 2 v glavnem pomnilniku vrednost v register v procesorju). Če želimo, da se instrukcija pravilno izvede morajo argumenti (naslov glavnega pomnilnika ali vrednost) obstajati v tako imenovanih **direktno dostopnih shranjevalnih napravah**. Če podatki, ki jih instrukcija zahteva niso v direktno dostopnih napravah jih predenj se instrukcija izvede moramo prenesti v direktno dostopne shranjevalne naprave.
Registri v procesorju so dosegljivi v enem CPE ciklu (2 GHz procesor --> ~ 0,5 ns).
Enako ne moremo reči za dostop do glavnega pomnilnika, saj se ta fizično nahaja zunaj procesorja in je s procesorjem povezan preko vodila. V primeru, da je podatek v glavnem pomnilniku je potrebno več ciklov, da se podatek prenese v registre procesorja in procesor zato čaka in njegovi cikli so neizkoriščeni. Rešitev je dodajanje procesorju **hitri pomnilnik** med procesorjem in glavnim pomnilnikom (**cache** – Pentium 4 imajo zato tudi do 1MB hitrega pomnilnika).
Relativno majhna hitrost dostopa do glavnega pomnilnika ni edini problem.
Zagotoviti moramo tudi, da je operacijski sistem v pomnilniku zaščiten pred uporabniškimi procesi v glavnem pomnilniku. Zaščito moramo implementirati s pomočjo strojne opreme.
Najprej moramo zagotoviti, da ima vsak proces ločen pomnilniški prostor in da proces lahko dostopa samo do svojega pomnilniškega prostora – do **veljavnih naslovov**. Zaščito lahko rešimo z dvema registroma: baznim in limitnim registrom.
Bazni register vsebuje najmanjši še veljaven naslov v glavnem pomnilniku za posamezen proces.
Limitni register vsebuje velikost naslovnega prostora v glavnem pomnilniku.
Če na primer vrednost baznega registra 30004 in limitnega registra 12090 potem bo proces domoval na naslovu od 30004 do vključno 42094.
Zaščita pomnilniškega prostora se izvaja tako, da strojna oprema v procesorju vedno primerja **vsak** tekoči naslov z vrednostjo baznega in limitnega registra. Vsak poizkus uporabniškega procesa da dostopa izven svojega pomnilniškega prostora izvede past operacijskemu sistemu in ta se pokaže kot **usodna napaka** (fatal error). Tak način preprečuje, da bi uporabniški proces (nehote ali zlonamerno) spreminjal vsebino operacijskega sistema ali drugega uporabniškega procesa.



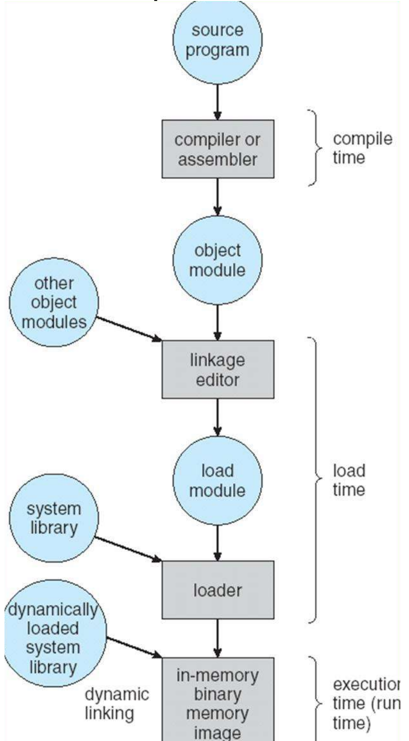
Dostop do spreminjanja vrednosti baznih in limitnih registrov ima **samo** operacijski sistem

preko **privilegiranih instrukcij,** ki se lahko izvedejo samo v sistemskem načinu dela!

6.1.3. Dodeljevanje absolutnih naslovov



Ponavadi je program shranjen na disku v binarni izvršljivi obliki (.exe). Ob zagonu programa se mora program preseliti v glavni pomnilnik, kjer se bo izvedel. Med izvajanjem programa se lahko proces prenaša iz virtualnega pomnilnika (npr. trdi disk) v glavni pomnilnik in obratno, odvisno od sheme upravljanja. Vrsti, ki čakajo na virtualnem pomnilniku (npr. disku) za izvajanje pravimo **vhodna vrsta**.
Po normalni proceduri bi se izbral en proces iz vhodne vrste in prenesel v glavni pomnilnik, kjer bi se začel izvajati. Med izvajanjem procesa bi procesor posegal po instrukcije in podatke, ki so med izvajanjem v pomnilniku. Ko bi se proces končal, bi naslovni prostor bil deklariran kot sproščen.



Slika prikazuje korake, preko katerih mora iti uporabniški proces predenj se začne izvajati v pomnilniku.
Naslovi so v izvornem programu (source program) običajno **simbolični** (npr. count).
Prevajalnik (compiler) spremeni simbolične naslove (jih na novo generira) v **relativne naslove** (relocatable addresses) (npr. "14 bytov od začetka modula").
Urejevalnik povezav in **nalagalnik** relativne naslove poveže z **absolutnimi naslovi** v pomnilniku (npr. relativni naslov "14 bytov od začetka modula" se prevede v absolutni naslov npr. 74014).
Dodeljevanje absolutnih naslovov je torej pretvorba naslovov iz enega tipa naslovov v drug tip naslovov.

Dodeljevanje naslovov instrukcijam in podatkom v pomnilniku (fizične naslove) se lahko zgodi v enem od naslednjih prizorišč:

- **času prevajanja** – če vemo v času prevajanja izvorne kode programa na kakšnem naslovu v pomnilniku bo program domoval, potem generiramo preveden program z že absolutnimi naslovi.
  Če se kasneje začetni naslov programa v pomnilniku spremeni je treba izvorni program ponovno prevesti, da dobimo nove absolutne naslove programa.
  MS-DOS .COM format programi vsebujejo kodo z absolutnimi naslovi ko jih prevedemo.
- **času nalaganja** – če v času prevajanja programa ne vemo kje v pomnilniku bo programska koda domovala, mora prevajalnik generirati novo kodo z relativnimi naslovi. V tem primeru je dodeljevanje naslovov dodatno zakasnjeno s časom nalaganja. Če se v

tem primeru spremeni začetni naslov, moramo samo ponovno naložiti programsko kodo, da se nam naslovni prostor prilagodi na nov začetni naslov.

- **času izvajanja** – če želimo, da proces med izvajanjem potuje iz enega naslovnega prostora v drugi naslovni prostor v pomnilniku. Če želimo, da to deluje moramo imeti podporo s strani strojne opreme (bazni in limitni register). Večina današnjih operacijskih sistemov uporablja to metodo.

6.1.3.1. Logični naslovni prostor – fizični naslovni prostor

Naslov, ki ga generira procesor imenujemo **logični naslov**, medtem ko naslov, ki ga vidi pomnilnik (naslov naložen v register glavnega pomnilnika (memory-address register -MAR)) je imenovan **fizični naslov**.
Logične naslove med izvajanjem procesa imenujemo tudi **virtualni naslovi**.

Dodelitev naslovov iz **logičnega naslovnega prostora** v **fizični naslovni prostor** v času izvajanja procesa poteka preko strojne naprave imenovane **enota za upravljanje pomnilnika** (memory-management unit – MMU).
Bazni register je tule imenovan **relativni register** (relocation register) in predstavlja začetni naslov. Vrednost relativnega registra se sešteje z naslovom, ki ga generira uporabniški proces ob času, ko hoče dostopati do pomnilniške lokacije.
Na primer, če je vrednost relativnega registra 14000, potem se bo, ko bo program hotel dostopati do logičnega naslova 0 iz programa, pretvoril v fizični naslov 14000 v pomnilniku. Torej naslov 0 v programu se dinamično pretvori v naslov 14000 v pomnilniku. Naslov 346 v programu se pretvori v naslov 14346 v pomnilniku.
Kot primer povejmo, da MS-DOS operacijski sistem na 80x86 procesorjih uporablja 4 relativne registre, ko nalaga in izvaja procese.
Uporabniški program nikoli ne vidi pravih - fizičnih naslovov.

6.1.3.2. Dinamično nalaganje in povezovanje

V prejšnji razlagi so bili program in podatki procesa vsi v fizičnem pomnilniku medtem ko so se izvajali. Da pridobimo boljši izkoristek pomnilniškega naslovnega prostora uporabimo **dinamično nalaganje** (dynamic loading). Po tem principu je v glavni pomnilnik naložena samo glavna rutina programa. Ostale rutine programa domujejo na virtualnem pomnilniku (npr. disku) in se naložijo v glavni pomnilnik samo takrat, ko jih glavna rutina pokliče.
Prednost dinamičnega nalaganja je v tem, da del programa, ki se ne izvaja tudi ne zaseda glavnega pomnilnika.
Dinamično nalaganje ne zahteva posebne podpore s strani operacijskega sistema. Dinamično nalaganje je odvisno od programerja ali bo tako metodo uporabil pri programiranju.
Dinamično povezovanje je podobno nalaganju, vendar se pri dinamičnem povezovanju v čas izvajanja namesto nalaganja rutin da **povezovanje do rutin** (npr. sistemske rutine

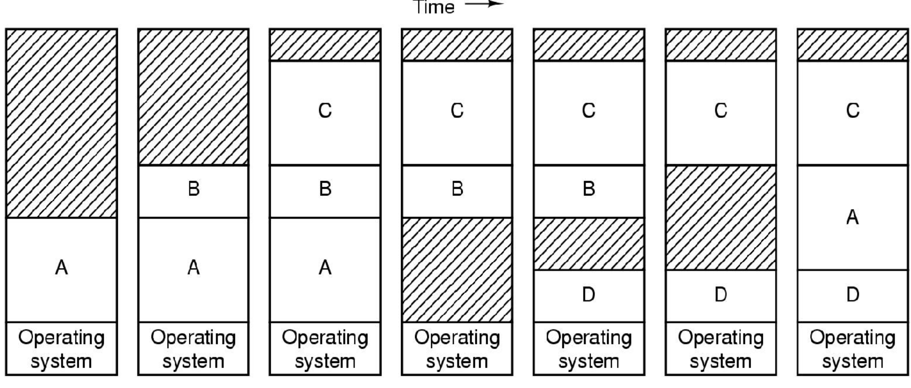
- knjižnice). Brez dinamičnega povezovanja, bi moral vsak program, ko se prevede imeti poleg osnovne kode tudi kopijo sistemske rutine, ki jo potrebuje, ko se izvaja. To pomeni, da bi vsak program zasedal več prostora v pomnilniku in disku, kot sicer z metodo dinamičnega povezovanja.
Pri dinamičnem povezovanju je v kodi dodan samo **štricelj** (stub). To je majhen kos kode, ki zna poiskati na katerem naslovu, kjer je naložena knjižnica, oz. jo zna naložiti v pomnilnik, če je v pomnilniku še ni.
Prednost dinamičnega povezovanja je tudi praktično odpravljanje napak (bug fixes). V tem primeru zamenjamo slabo knjižnico z popravljeno in programska koda uporablja popravljeno knjižnico.
Uporaba dinamičnega povezovanja zahteva pomoč od operacijskega sistema. Če upoštevamo dejstvo, da je pomnilniški prostor zaščiten eden od drugega je potem samo operacijski sistem tisti, ki lahko pogleda ali je v pomnilniku že naložena knjižnica, ki jo potrebuje program oz. če lahko do pomnilniškega prostora, kjer je knjižnica lahko dostopa več procesov. Nekateri operacijski sistemi podpirajo samo dinamično nalaganje.

6.1.3.3. Začasno izločanje (swapping)

Proces mora biti v pomnilniku, če se hoče izvesti. Proces lahko zamenja svoje mesto, tako da se začasno izloči iz pomnilnika na navidezni pomnilnik (npr. disk).
Normalno se proces, ki je bil začasno izločen na disk zopet preseli nazaj v pomnilnik na enak naslovni prostor. To je seveda odvisno od tega kdaj se naslovni prostor pretvarja.
Če se pretvarjanje naslovov izvrši med prevajanjem, potem je nujno, da se naslovni prostor ne sme zamenjati med zamenjavo.
Če se pretvarjanje naslovov izvrši med izvajanjem, potem ni nujno, da se med zamenjavo ohrani enak naslovno prostor, saj se fizični naslovi izračunavajo in prilagajajo med izvajanjem.

Navidezni pomnilnik mora biti čimbolj hiter (pri Linuxih, kjer je eden izmed razdelkov obvezno virtualni razdelek (swap partition), in ga določimo nekje na sredini diska, ker je dostopni čas na sredini diska najmanjši). Navidezni pomnilnik mora biti dovolj velik, da lahko shrani kopije vseh uporabniških procesov (pri Linuxu je običajno virtualni razdelek velik 2* velikost delovnega pomnilnika).
Sistem vzdržuje pripravljeno vrsto v kateri so pripravljeni procesi, katerih naslovni prostori so v delovnem pomnilniku ali na navideznem pomnilniku. Ko se razvrščevalnik procesorja odloči, da bo izvedel proces najprej pokliče dodeljevalnik (dispatcher). Dodeljevalnik preveri kje je proces, ki ga hoče procesor izvesti. Če ga ni v glavnem pomniku in niti ni prostora v glavnem pomnilniku, potem dodeljevalnik zamenja trenutni proces v pomnilniku s tistim, ki ga procesor želi. Potem ponovno naloži vse registre in preda kontrolo procesu.

Čas za menjavo okolja (context switch) je pri zamenjavi dolg. Zamislimo si uporabniški proces, ki zasede 1 MB in navidezni pomnilnik je trdi disk s hitrostjo prenosa 5 MB/s. Prenos takega procesa v ali iz pomnilnika bi trajal 200 ms. Pri zamenjavi procesa bi trajalo 400 ms. Pri Round-Robin metodi razvrščanja na procesorju bi pomenilo, da bi moral biti cikel daljši od 400 ms.



6.1.4. Nedeljivo dodeljevanje procesa v pomnilnik

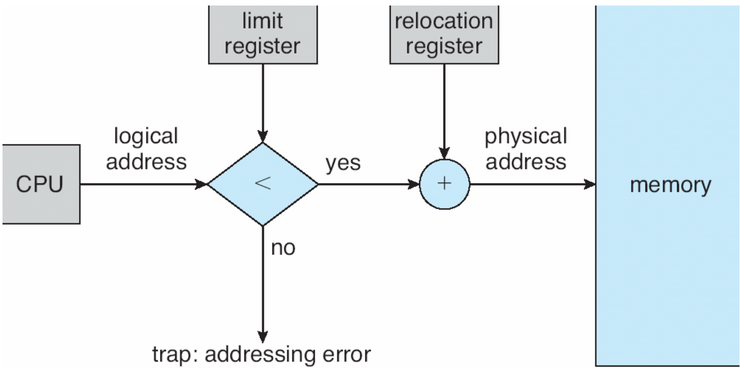
Običajno je pomnilnik razdeljen na dva razdelka:

- razdelek operacijskega sistema, ki konstantno domuje na svojem naslovnem prostoru
- razdelek uporabniških procesov, kjer je dogajanje dinamično

Pri nedeljivem naslavljanju pomnilniškega prostora imamo procese, ki zasedajo pomnilniški prostor v enem kosu – nedeljivo. Premisliti moramo, kako bomo dodeljevali prost pomnilniški prostor uporabniškim procesom, ki čakajo v vhodni vrsti.

6.1.4.1. Zaščita pomnilniškega prostora

Predenj povemo kaj več o naslavljanju moramo povedati nekaj o zaščiti pomnilniškega prostora med operacijskim sistemom in uporabniškim naslovnim prostorom.
To storimo lahko z relativnim registrom (relocation register), kot smo že prej opisali. Relativni register vsebuje najnižji fizični naslov pomnilnika, limitni register pa obseg logičnih naslovov (npr. relativni register = 100040 in limitni register 74600). Strojna enota MMU mapira logične naslove dinamično s tem, da doda relativnemu registru vrednost limitnega registra. (nova vrednost relativnega registra = 174640).
Ko razvrščevalnik procesorja izbere proces za izvajanje, dodeljevalnik naloži relativni in limitni register s pravimi vrednostmi, kot del menjave okolja. Ker se vsak logični naslov generiran iz procesorja preveri s temi registri ni mogoče, da bi tekoč proces posegel po naslovnem prostoru operacijskega sistema ali pomnilniškem prostoru drugega uporabniškega procesa.
S pomočjo relokacijskega registra je mogoče efektivno doseči, da se naslovni prostor, ki ga zaseda operacijski sistem dinamično spreminja. Spreminjanje naslovnega prostora operacijskega sistema je ugodno v več primerih. Operacijski sistem vsebuje kodo in registerski prostor za gonilnike naprav. Če se naprava ne uporablja, je odveč držati



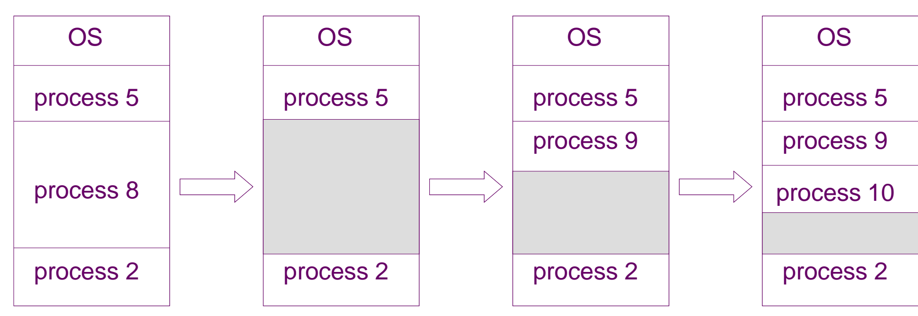
gonilnik v pomnilniku. Raje bi prostor izkoristili za kaj drugega. Taka koda se imenuje **tranzitna koda** operacijskega sistema. Z uporabo te kode lahko zmanjšamo velikost operacijskega sistema med izvajanjem.

6.1.4.2. Naslavljanje pomnilnika

Najbolj enostavna metoda je razdeliti pomnilnik v **razdelke enake dolžine**. Vsak razdelek lahko vsebuje največ en proces. Stopnja večopravilnosti je tako odvisna od števila razdelkov. V tej metodi z več razdelki se proces, ki je na vrsti naloži iz vhodne vrste v pomnilnik in čaka na izvajanje. Ko se izvede do konca se razdelek sprosti. Take metode se danes ne uporablja več.
Druga metoda:
Operacijski sistem vzdržuje tabelo v kateri piše, kateri del pomnilnika je prost in kateri je zaseden. Na začetku imamo na razpolago ves pomnilnik namenjen uporabniškim procesom kot en velik blok pomnilnika imenovan tudi **luknja**. Ko proces pride in potrebuje pomnilnik, poiščemo dovolj veliko luknjo v pomnilniku, v katero lahko stlačimo proces. Ko luknjo dobimo zasedemo točno toliko naslovnega prostora kolikor ga potrebujemo. Ko je proces v pomnilniku lahko tekmuje za izvajanje na procesorju, seveda po pravilih razvrščevalnika.
V vsakem trenutku imamo seznam prostih blokov in vhodno vrsto. Operacijski sistem lahko razvrsti vhodno vrsto glede na razvrščevalni algoritem. Pomnilnik je zaseden z uporabniškimi procesi dokler se ne pojavi nov proces, ki potrebuje pomnilnik in v njem ni več luknje, ki bi procesu zadoščala. V tem primeru operacijski sistem počaka, da se pojavi dovolj velika luknja ali pogleda v vhodno vrsto, če je še kakšen manjši proces, ki pa bi ga lahko prenesel v pomnilnik.
S takim naslavljanjem pomnilniškega prostora pridemo do situacije, ko imamo v pomnilniku **niz lukenj** razmetanih po vsem pomnilniku. Ko proces pride in želi preiti v pomnilnik, sistem pogleda po nizu lukenj in proces dodeli luknji, ki je dovolj velika. Če je luknja prevelika jo sistem zazdeli na dva dela:

- naslovni prostor za proces in
- prostor, ki spada v nov niz lukenj

Ko se proces zaključi se luknja sprosti in preide zopet v niz lukenj. Če pride do situacije, ko sta dve luknji sosedi, pride do združitve dveh lukenj v eno večjo luknjo in mogoče se pa sedaj dobi proces, ki čaka na pomnilnik.



Taka procedura je tipičen primer problema, ki se pojavi pri **dinamičnem dodeljevanju pomnilniškega prostora**. Kako dodeliti prostor procesu z velikostjo n preko seznama prostih lukenj.
Ta problem rešujemo na več načinov.
Eden je ta, da pregledamo luknje in na podlagi seznama velikosti lukenj se odločimo v

katero luknjo bomo postavili proces. Poznamo naslednje strategije iskanja lukenj:

- **prvo ujemanje (First-Fit)** - ko naleti na prvo primerno veliko luknjo vanjo naloži proces. Je nahitrejše.
- **najboljše ujemanje (Best-Fit)** - poišče luknjo, ki je najprimernejša za proces (velikost luknje se najbolje ujema z velikostjo procesa). Iskati mora po vseh luknjah, razen če so luknje že razporejene po velikosti.
- **najslabše ujemanje (Worst-Fit)** – poišče največjo luknjo med luknjami in ji dodeli proces. Tule ostane velik del luknje še za druge procese.

Pri naštetih algoritmih pride prej ali slej do **zunanje drobitve** (external fragmentation). Medtem ko se procesi sprehajajo v in iz pomnilnika se pomnilniški prostor drobi na majhne dele pomnilnika. **Zunanja drobitev** obstaja, ko imamo na razpolago dovolj prostora za naložit naslednji proces, vendar tak prostor ni v enem kosu, ampak je razdrobljen po celem pomnilniku – pomnilniški prostor je razdrobljen na več majhnih lukenj.
Statistični podatki kažejo, da pri strategiji "prvo ujemanje" se pri n procesih dodeljenih v pomnilniku ustvari drobitev v velikosti 0,5*n procesov (npr. če smo po strategiji "prvo ujemanje" spravili 10 procesov v pomnilnik smo ustvarili drobitev veliko 5 procesov).

6.1.5. Drobljenje (fragmentation)

Drobljenje pomnilniškega prostora je lahko zunanje ali notranje.
Zunanje drobljenje smo prikazali prej, kjer se znotraj luknje pojavi majhna luknja. Na primer imamo luknjo veliko 18464 bytov in proces velik 18462 bytov. Ko sistem zasede luknjo od luknje ostane luknjica velika 2 byta. Program, ki vzdržuje informacijo o luknji, bi bil večji kot luknja sama.
Notranje drobljenje lahko prikažemo pri metodi dodeljevanja pomnilnika, kjer je pomnilnik razdeljen na razdelke (bloke) s fiksno velikostjo. Pomnilnik se dodeljuje glede na velikost bloka. Tukaj se lahko zgodi, da je velikost procesa, ki želi pomnilnik manjša kot velikost fiksnega bloka pomnilnika. Razlika v teh dveh številkah nam poda velikost notranje razdrobljenosti – pomnilniški prostor, ki je znotraj razdelka in ni uporabljen.

Zunanje drobljenje lahko rešimo z **združevanjem** (compaction). Cilj te rešitve je v tem, da se vse majhne luknje združijo v večje luknje. Združevanje ni možno v primerih ko se nam naslovni prostor prevede med prevajanjem ali med nalaganjem procesa (čas prevajanja ali čas nalaganja). Možno je samo, če imamo dinamično dodeljevanje naslovov, ki se izvede med izvajanjem procesa. Naslovni prostor procesa se premakne na drugi naslovni prostor in tako naredi prazen prostor večji.
Druga rešitev predpostavlja, da dovolimo logičnemu naslovnemu prostoru, da ni ves v enem kosu, ampak se razdeli na naslovne prostore, ki so v pomnilniku nezasedeni. Tako reševaje problema zunanjega drobljenja predstavljata dve komplementarni tehniki:

- **stranjenje** (paging) in
- **segmentacija** (segmentation)

6.1.6. Stranjenje (paging)

Stranjenje je shema upravljanja z pomnilnikom, ki dovoli fizičnemu naslovnemu prostoru procesa, da ni v enem kosu. Tako shemo uporablja večina modernih operacijskih sistemov. Zadnje implementacije stranjenja ponujajo tesno sodelovanje operacijskega sistema in strojne opreme, še posebej na 64 bitnih mikroprocesorjih.

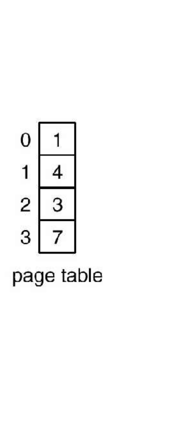
6.1.6.1. Osnovna metoda

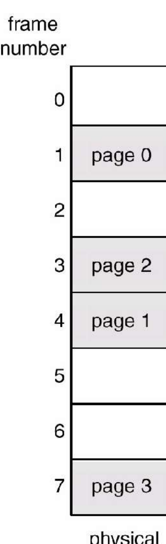
Fizični pomnilnik je razdeljen na **bloke s fiksno dolžino** imenovane **okvirji** (frames) (od 512 bytov – 8192 bytov).
Navidezni pomnilnik (npr. disk) je tudi razdeljen na bloke enake dolžine imenovane **strani** (pages).
Velikost blokov v fizičnem pomnilniku in navideznem pomnilniku je enaka.
Ko pride proces, se njegove strani naložijo iz navideznega pomnilnika v prazne bloke (frames) v fizičnem pomnilniku. Strojna podpora metodi stranjevanja kaže slika.

Vsak naslov, ki ga generira procesor se razdeli na dva dela:

- **številko strani** (page number) – p
- **odmik strani** (page offset) – d

Številka strani določa indeks v **tabeli strani** (page table). Tabela strani vsebuje bazne naslove (f) vsakega okvirja v fizičnem pomnilniku. Bazni naslov se v kombinaciji z odmikom strani preslikajo v fizični naslov, ki se posreduje fizičnemu pomnilniku.
Model prikazuje naslednja risba.
Velikost strani kot velikost bloka v fizičnem pomnilniku je definirana preko strojne opreme in je ponavadi potenca števila 2 (npr. 2^9=512 B do velikosti 16 MB). Izbira potence z osnovo 2 kot velikost strani daje dokaj enostavno pretvorbo logičnega naslova v številko strani in odmik strani.
(npr. če imamo logični naslovni prostor velik 2^m bytov in velikost strani 2^n naslovnih enot ali bytov potem je številka strani enaka najvišjim (m-n) bitom logičnega naslova in odmik strani enak najnižjim n bitom logičnega naslova.





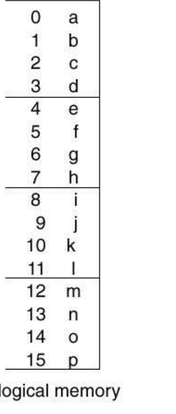
Primer:

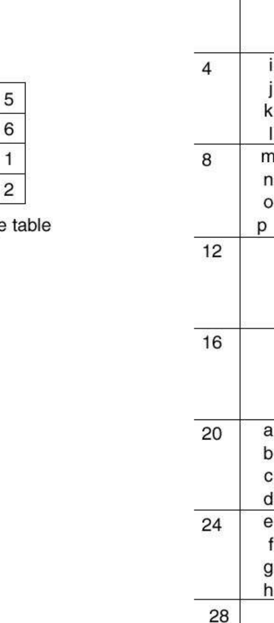
logični naslov	št. strani(p)	odmik strani(d)
12=1100$_{(2)}$	11$_{(2)}$=3 stran	00$_{(2)}$=0 odmik
zgornjih (m-n) bitov		spodnjih n bitov

Konkreten primer, ki uporablja velikost strani=velikost okvirja 4 B in fizični pomnilnik velik 32 B (8 strani).

Logični naslov 0 pripada strani številka 0 in odmiku 0. Če pogledamo v tabelo strani vidimo, da strani 0 pripada 5 okvirju v fizičnem pomnilniku.

- logični naslov 0 (stran 0, odmik 0), ki ga generira procesor se pretvori v fizični naslov 20 v fizičnem pomnilniku (4*5+0=20)
- logični naslov 3 (stran 0, odmik 3), ki ga generira procesor se pretvori v fizični naslov 23 (4*5+3)
- logični naslov 4 (stran 1, odmik 0), ki ga generira procesor se pretvori v fizični naslov 24 (4*6+0)





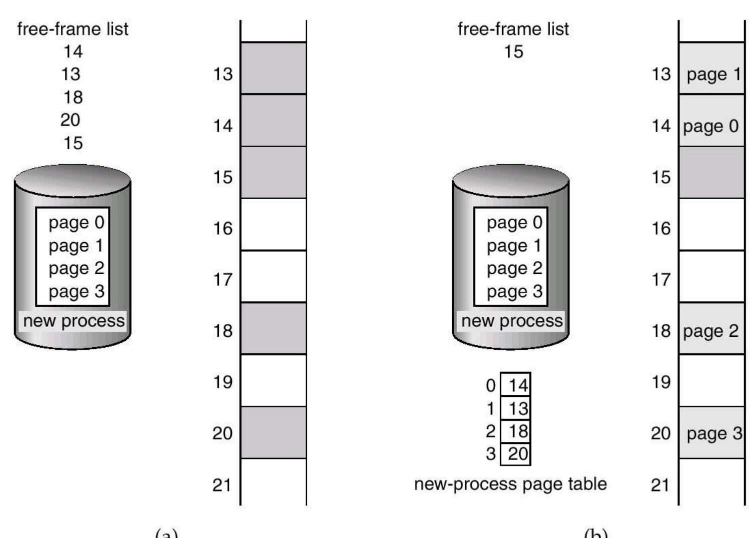
Ko uporabljamo stranjenje nimamo zunanjega drobljenja, vsak prosti okvir je lahko zapolnjen s stranmi procesa.

Lahko pa imamo **notranje drobljenje**. Če se velikost procesa ne ujema z faktorjem velikosti strani se na zadnji strani pojavi neizkoriščen prostor (npr. imamo strani dolge 2^{11}=2048 B in proces velik 72766 B, potrebovali bi 72766/2048 =35 strani in ostanek 1086 B, kar bi zapolnilo približno polovico 36-te strani). Takemu procesu bi bilo dodeljenih 36 strani z notranjim drobljenjem (2048-1086=) 962 B. V najslabšem primeru bi proces zasedel n število strani + 1B na naslednji strani, kar bi pomenilo drobljenje veliko skoraj eno celo stran.

Torej, če imamo velikost procesov neodvisnih od velikosti strani se notranjemu drobljenju ne moremo izogniti in je v povprečju veliko 0,5 strani.

Veliko manjše drobljenje dobimo, če imamo strani čim manjše. Seveda na drugi strani pa se tabela strani povečuje z manjšanjem velikosti strani in ne smemo pozabiti, da prenos pomnilnik-trdi disk in obratno hitreje prenaša večje strani kot več manjših. V osnovi se je velikost strani povečevala s zmogljivostjo procesorja, podatkovnih nizov in glavnega pomnilnika. Danes so strani velike od 4 kB do 8kB, nekateri sistemi podpirajo tudi več velikosti strani (Solaris uporablja 8kB in 4MB stran, odvisno kakšni podatki se predstavljajo preko strani). Raziskovalci se sedaj ukvarjajo z razvojem podpore za aktivno spremenljive velikosti strani.

Ko proces prispe v sistem, da bi se izvedel se preveri njegova velikost izražena v straneh. Vsaka stran procesa potrebuje en blok v fizičnem pomnilniku. Če proces zahteva n strani najmanj n blokov mora biti prostih v fizičnem pomnilniku. Če je n blokov prostih se te bloke dodeli procesu. Prva stran procesa se naloži v prazen blok in številka bloka se shrani v tabelo strani, itd.. Na sliki vidimo proste bloke pred dodeljevanjem (a) in po dodeljevanju (b).

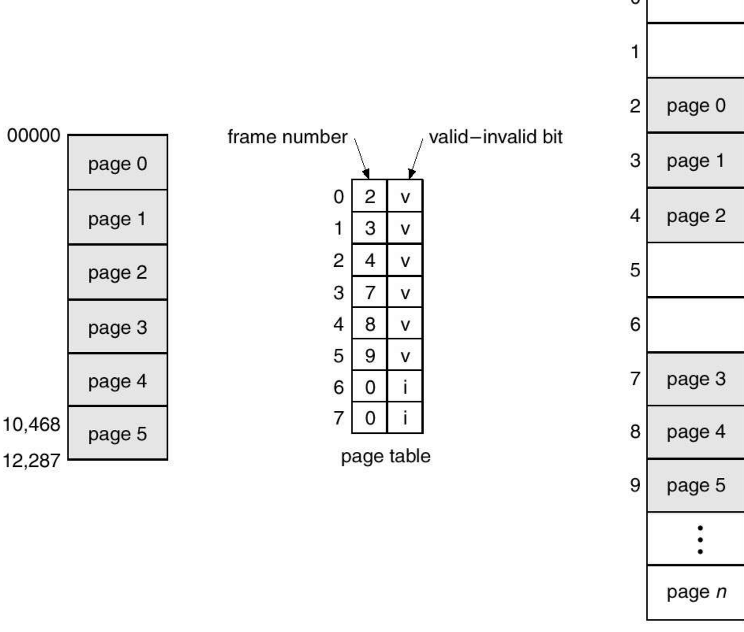


6.1.6.2. Zaščita

Zaščita pomnilnika v okolju s stranmi je izveden z **zaščitnim bitom**. Bit je dodan tabeli strani. S tem bitom definiramo ali v posamezno stran lahko beremo in pišemo (read/write) ali samo beremo (read only). Poizkus pisanja v zaščiteno (read only) stran povzroči strojno past (trap) operacijskemu sistemu (ali kršitev zaščitenega pomnilniškega prostora (memory-protection violation).

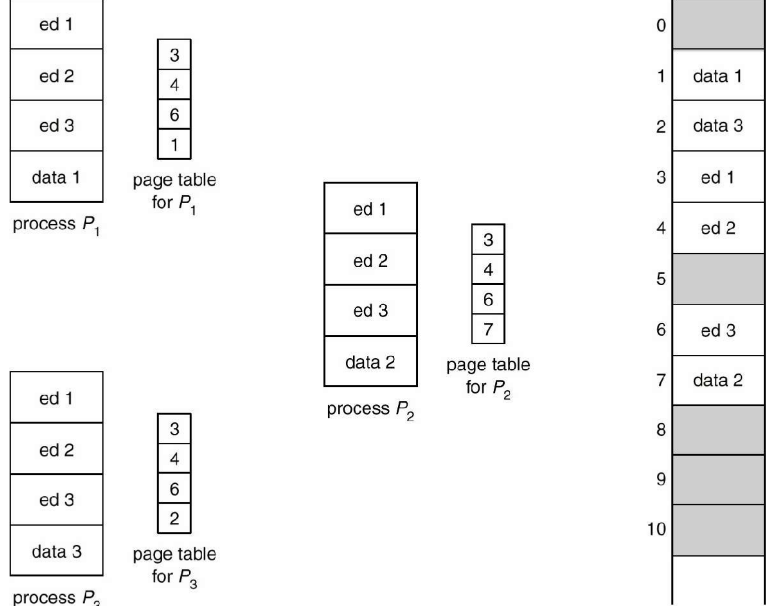
Tabeli strani je dodan še en bit (veljaven-neveljaven). Njegove dve vrednosti povesta če je pripadajoča stran v logičnem naslovnem področju procesa ali ne. Neveljavni naslovi se tako ulovijo s tem bitom v past.

Na primer v sistemu s 14 bitnim naslovnim prostorom, kjer lahko naslovimo (2^{14}=16384 naslovov – od 0 do 16383) imamo program, ki zahteva samo naslove od 0 do 10468. Pri velikosti strani 2KB imamo program, ki zasede 5 strani in 228 naslovov 6-te strani. Naslovi generirani preko procesorja, ki spadajo v logični naslovni prostor strani od 0 do 5 se uspešno alocirajo v fizični pomnilnik, ker je bit postavljen na veljaven (v). Vsak poizkus generiranja logičnega naslova, ki pripada naslovnemu prostoru 6-te ali 7-me strani (i) bi povzročil past operacijskemu sistemu (neveljavna referenca strani (invalid page reference).



6.1.6.3. Strani v skupni rabi

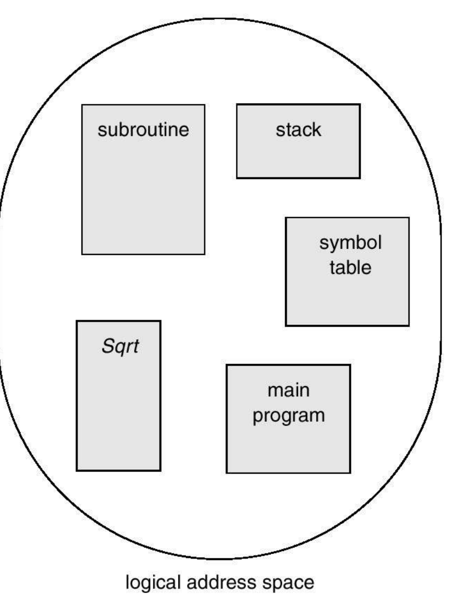
Še ena prednost stranjenja je možnost skupne rabe kode programa. Zamislimo si sistem, ki podpira 40 uporabnikov in vsak poganja svoj urejevalnik teksta. Če urejevalnik zasede 150 KB kode in 50 KB prostora za podatke, bi potrebovali 8000 KB za vzdrževanje 40 uporabnikov. Na sliki vidimo 3 procese dolge po 4 strani (od tega 3 strani za kodo programa in 1 stran za podatke). Koda programa-urejevalnika v fizičnem pomnilniku pa zaseda za vse 3 procese samo 3 bloke. Kodo programa si torej uporabniki delijo. Koda podatkov pa je za vsak proces posebej naslovljena v fizičnem pomnilniku. 40 uporabnikov, ki poganjajo urejevalnik zaseda 3 strani za kodo programa in 40 strani za podatke, kar znese (40+3)*50KB=2150KB. Veliko manj, kot če kode programa, ki je med izvajanjem nespreminjajoča (reentrant) nebi delili vsi uporabniki.



6.1.7. Segmentacija

6.1.7.1. Osnovna metoda

Kako programer misli o programu, ko ga piše? Ko programer piše program razmišlja o glavnem programu s nizom podrutin, procedur, funkcij ali modulov. Lahko tudi razmišlja o podatkovnih strukturah kot so tabele, polja, skladi, spremenljivke, ... Pogovarja se o modulih kot so tabele s simboli, funkcija sqrt, glavni program, brez da bi ga zanimalo katere naslove v fizičnem pomnilniku našteti moduli zasedajo. Vsak naštet **segment** zasede različne dolžine – dolžina segmenta je odvisna samo od namena segmenta v programu. Elementi znotraj posameznega segmenta so predstavljeni z odmikom od začetka segmenta (prvi stavek programa, 17 vnos v tabeli simbolov, 5 instrukcija v funkciji "sqrt", itd.). Segmentacija je shema upravljanja s pomnilnikom, ki podpira uporabniški pogled na pomnilnik. Logični naslovni prostor je kolekcija **segmentov**. Vsak segment vsebuje ime in dolžino. Naslov je določen z **imenom segmenta** in **odmikom znotraj segmenta**. Uporabnik določi naslov z dvema parametroma: ime segmenta in odmik (spomnimo, da pri stranjenju je uporabnik določil logični naslov, ki se je preko strojne opreme preslikal v številko strani in odmik, vse skupaj uporabniku nevidno). Za

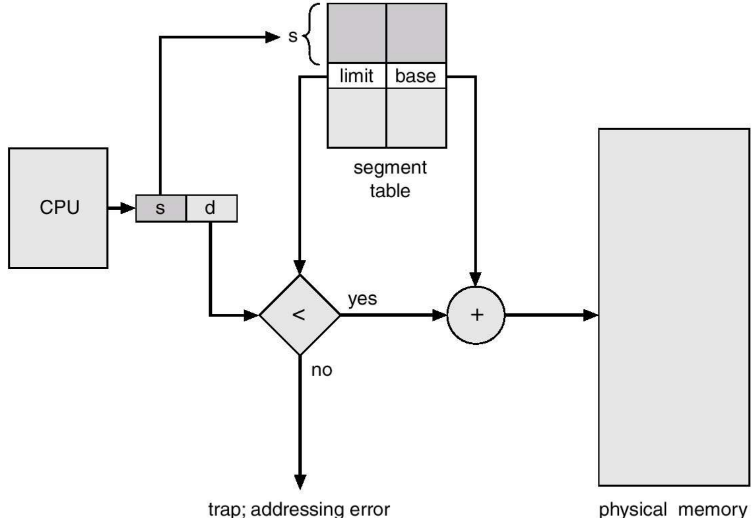


poenostavitev ima vsak segment namesto imena številko.
Logični naslov zgleda takole: **<številka-segmenta, odmik>**.
Normalno se uporabniški program prevede in prevajalnik avtomatsko zgradi segmente, ki so rezultat napisanega programa (pascalski prevajalnik lahko naredi ločene segmente za globalne spremenljivke, lokalne spremenljivke vsake procedure in funkcije, del kode, ki pripada vsaki proceduri in funkciji in sklad klica procedure). Nalagalnik bi sprejel vse segmente in jim dodelil številko-segmenta.

6.1.7.2. Strojna oprema

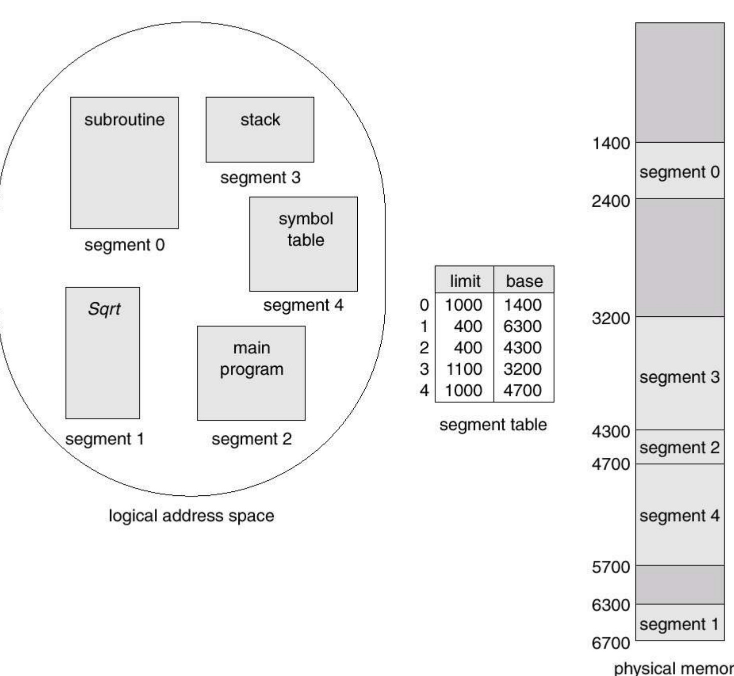
Uporabnik sedaj lahko naslovi objekte v svojem programu preko dvodimenzionalnega naslova (<številka-segmenta, odmik>). Na strani fizičnega pomnilnika pa velja, da imamo še vedno eno dimenzijski niz naslovov. Torej moramo implementirati rešitev, ki bo preslikala dvodeimenzijski logični naslovni prostor v enodomenzijski fizični naslovni prostor. Preslikava je implementirana preko **segmentne tabele**.
Vsak vnos v segmentni tabeli ima:

- **bazni segment** – vsebuje začetni fizični naslov, segmenta, ki leži v fizičnem pomnilniku in
- **limitni segment** – določa dolžino segmenta.



Logični naslov pa vsebuje številko segmenta (s) in odmik v segmentu (d). Številka segmenta pomeni indeks v segmentni tabeli.

Kot primer si poglejmo situacijo na sliki.
Imamo 5 segmentov (številke segmentov od 0 do 5). Segmenti so shranjeni v fizičnem pomnilniku kot kaže slika. Segmentna tabela ima ločen vnos za vsak segment (bazno vrednost in limitno vrednost). Na primer segment 2 je dolg – limitiran na 400 B in se začne na baznem naslovu 4300. Tako bi se logičen naslov 53 v segmentu 3 preslikal v fizičen naslov 4353 (4300+53) v fizičnem pomnilniku, itd..



6.2. Vprašanja in naloge za ponavljanje

1. Do katerih shranjevalnih struktur lahko procesor dostopa **direktno**?

2. Kako deluje zaščita pomnilniškega prostora procesa?

3. Kakšna je razlika med logičnim in fizičnim naslovnim prostorom?

4. Kakšna je razlika med zunanjo in notranjo drobitvijo (fragmentacijo)?

5. Naštej tri dodeljevalne algoritme, ki dodelijo procesu prazen del pomnilnika. Kako bi vsak od treh algoritmov zasedal prazne razdelke, če imamo prazne razdelke v pomnilniku po vrsti: 100KB, 500KB, 200KB, 300KB in 600KB, in procese po vrsti velikosti: 212KB, 417KB, 112KB in 426KB. Kateri dodeljevalni algoritem najbolje izkoristi pomnilnik?

6. Imamo logični naslovni prostor dolg 8 B in fizični naslovni prostor dolg 32 B. V kateri fizični naslov se bo dodelil logični naslov št. 5, če je velikost strani 2 B in upoštevamo spodnjo tabelo strani.

0	12
1	4
2	7
3	14

7. Računalniški sistem ima dolžino strani 4KB, in 16b logični naslovni prostor. Koliko je lahko največja velikost tabele strani?
   Če je v fizičnem pomnilniku 128 okvirjev, koliko je velikost fizičnega naslovnega prostora?

8. Imamo naslednjo tabelo segmentov. Kakšni so fizični naslovi naslednjih logičnih naslovov: 0,430 - _____
   1,10 - _____
   2,500 - _____
   3,400 - _____
   4,112 - _____

Segment	Baza	Dolžina
0	219	600
1	2300	14
2	90	100
3	1327	580
4	1952	96

9. Kakšna je glavna razlika med stranjenjem in segmentacijo?

6.3. Virtualni pomnilnik

V prejšnjem poglavju smo govorili o strategijah upravljanja s pomnilnikom. Vse strategije imajo samo en cilj, to je držati več procesov istočasno v pomnilniku, da se omogoči večopravilnost. Omenjene strategije se nagibajo k temu, da je v fizičnem pomnilniku celoten proces predenj se proces začne izvajati. **Virtualni pomnilnik** pa je tehnika, ki omogoča izvajanje procesa, četudi ni v celoti v fizičnem pomnilniku. Prednost take tehnike je v tem, da so lahko programi, ki jih napiše programer daljši od fizičnega pomnilnika in se pri tem programer ne ukvarja z omejitvijo velikosti njegovega programa ampak z delovanjem in namenom programa. Virtualni pomnilnik omogoča procesom da enostavno delijo datoteke in naslovne prostore, ter priskrbi učinkovit mehanizem za ustvarjanje procesov.

6.3.1. Ozadje

Prej omenjeni algoritmi upravljanja s pomnilnikom so potrebni, ker instrukcija mora biti v fizičnem pomnilniku predenj se izvede. Prva rešitev je v tem, da naložimo celoten logični naslovni prostor v fizični pomnilnik, vendar, če pogledamo realne programe lahko vidimo, da celoten program ni potreben:

- programi imajo tudi kodo, v kateri je opisano, kako postopati v primeru napak. Ker so napake redke ali jih sploh ni se ta del kode lahko tudi nikoli ne izvede
- določene opcije programa se uporabijo zelo poredko (kolikokrat ste pri urejanju besedila uporabili pomoč, ki jo nudi urejevalnik besedil?, ...)

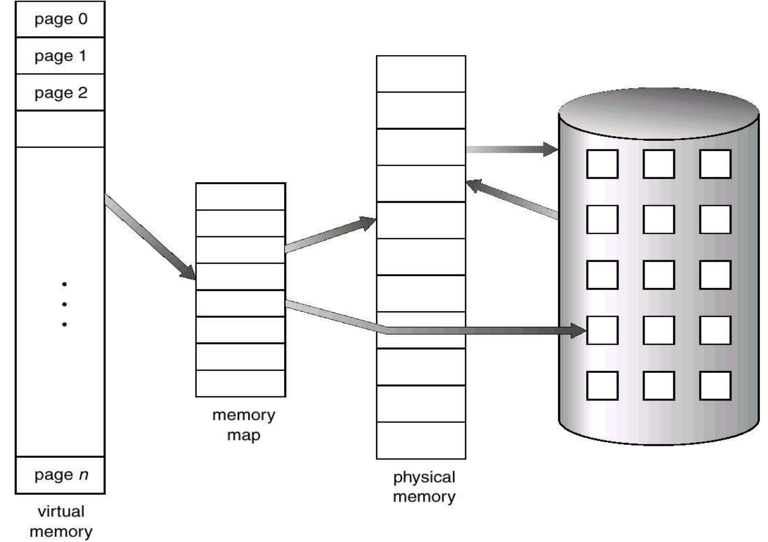
Zmožnost, da je program le delno v pomnilniku rodi veliko prednosti:

- program ne bo več stisnjen zaradi omejene velikosti fizičnega pomnilnika in programerji lahko pišejo programe z velikim logičnim naslovnim prostorom.
- ker je vsak uporabniški program le delno v fizičnem pomnilniku, ostane prostor še za druge programe in tako večja izraba procesorja in večja odzivnost sistema
- manj V/I operacij, ki naložijo ali zamenjajo program, programi bi tekli hitreje

S poganjanjem programa, ki ni v celoti v fizičnem pomnilniku pridobi tako sistem kot uporabnik.

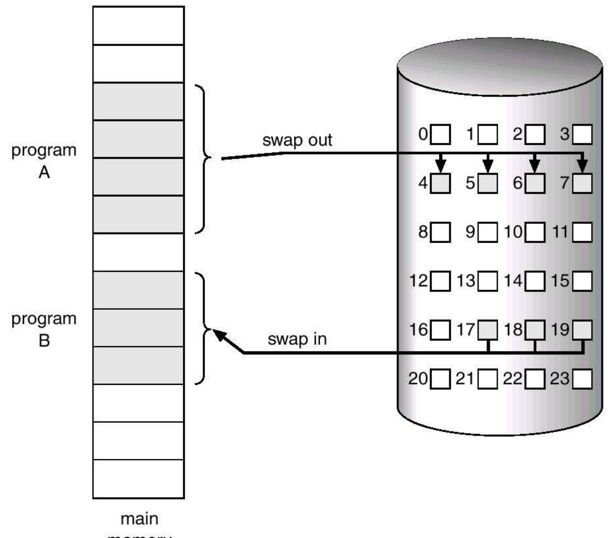
Virtualni pomnilnik je ločitev uporabniškega logičnega naslovnega prostora od fizičnega naslovnega prostora. Ta ločitev dovoljuje uporabo velikega logičnega prostora za programerje, kjer je fizični pomnilnik majhen v primerjavi z virtualnim.

Virtualni pomnilnik je lahko implementiran s **stranjenjem na zahtevo** ali s **segmentacijo na zahtevo**.



6.3.2. Stranjenje na zahtevo (demand paging)

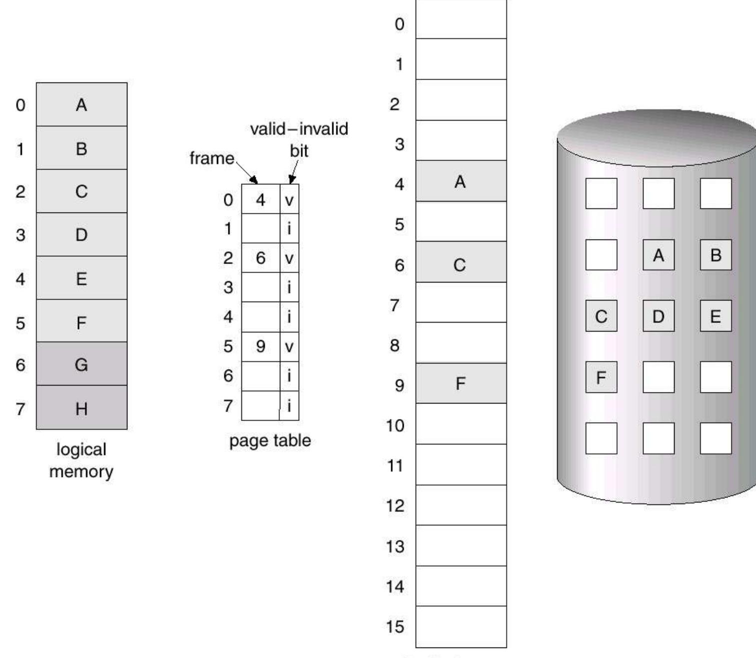
Stranjenje na zahtevo je podobno stranjenju z zamenjavo (glej sliko). Proces je normalno shranjen na sekundarnem pomnilniku (npr. disku). Ko želimo izvesti proces ga prenesemo iz sekundarnega pomnilnika v fizični pomnilnik. Namesto, da prenesemo celoten proces v fizični pomnilnik, uporabimo tako imenovanega **lenega menjalca**. Leni menjalec nikoli ne prenese v fizični pomnilnik strani, katere proces ne potrebuje. Prenesel jo bo samo takrat ko jo bo proces potreboval pri izvajanju. Sedaj gledamo proces kot niz strani ne pa kot en nedeljiv naslovni prostor. Namesto menjalec (swapper), ki zamenja ali naloži celoten proces bi pri stranjenju na zahtevo raje govorili o **stranilcu** (pager), ki skrbi za zamenjavo ali nalaganje strani.



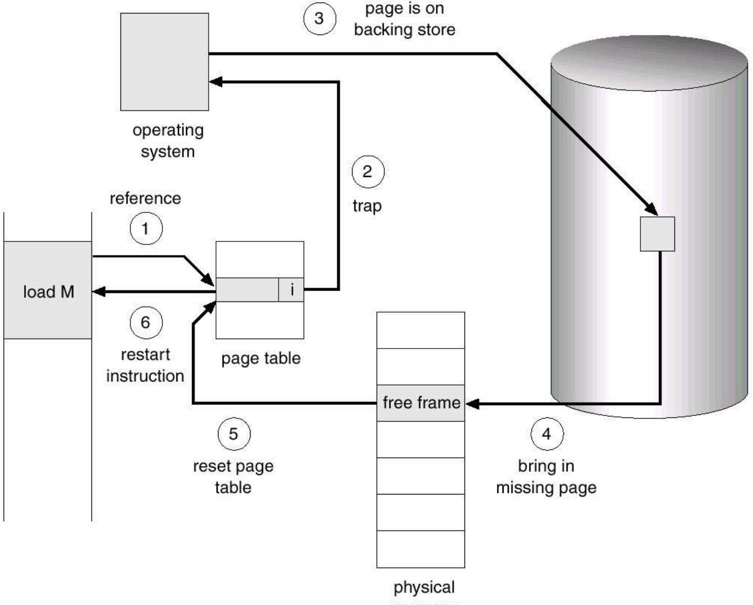
6.3.2.1. Osnovni koncept

Ko proces želi izvajanje, stranilec izbere in naloži samo strani, ki so potrebne za izvajanje procesa. Če želimo razlikovati med stranmi, ki so na disku in bloki, ki so v fizičnem pomnilniku potrebujemo strojno podporo. Lahko se uporabi shemo z veljavnim in neveljavnim bitom, ki smo jo omenili v prejšnjem poglavju. Ko je bit postavljen na veljaven (v) pomeni, da je stran pravilno naslovljena in da je v glavnem pomnilniku. Ko je bit postavljen na neveljaven (i) pomeni, da ali stran ni veljavna in da ni v logičnem naslovnem prostoru ali da je stran veljavna ampak še vedno na disku. Vnos v tabeli strani za stran, ki je prenesena v fizični pomnilnik je vedno enak, vnos v tabeli strani za stran, ki ni še v fizičnem pomnilniku pa je označena kot neveljavna (i) ali vsebuje naslov strani, ki je še na disku. Glej sliko.

Če je neka stran na disku in je v tabeli označena kot neveljavna (i), se ne bo zgodilo nič, dokler proces ne bo zahteval te strani. Ko ima proces v fizičnem pomnilniku stran, ki jo potrebuje, se proces normalno izvaja. Kaj se pa zgodi, če proces zahteva stran, ki je ni v fizičnem pomnilniku? Dostop do strani, ki je ni v fizičnem pomnilniku (bit je postavljen na neveljavno) povzroči **past** imenovano **napaka strani** (page-fault trap).



Odpravljanje napake strani vidimo v naslednjih korakih:



1. Najprej preverimo notranjo tabelo procesa (ponavadi v kontrolnem bloku procesa – PCB), kjer preverimo, ali je bil dostop do fizičnega pomnilnika veljaven ali neveljaven
2. Če je bil naslov s katerim smo dostopali do fizičnega pomnilnika neveljaven, se proces konča. Če je bil naslov veljaven, pomeni, da je stran še na disku
3. Poiščemo prosti blok (s tem, da vzamemo enega iz seznama prostih blokov)
4. Damo v vrsto proces, ki bo izvedel operacijo prenosa želene strani iz diska v nov alociran blok
5. Ko je prenos iz diska v fizični pomnilnik končan spremenimo notranjo tabelo procesa in tabelo strani tako, da zapišemo, da je stran v bloku fizičnega pomnilnika
6. Ponovno zaženemo instrukcijo, ki je zahtevala prej neveljavno stran. Sedaj seveda nebi smelo priti do napake strani in procesu se zdi, kot, da stran ni nikoli manjkala

6.3.3. Kreiranje procesov

Poznamo dve tehniki, ki jih podpira navidezni pomnilnik in izboljša učinkovitost kreiranja in izvajanja procesa. To sta **kopija ob zapisu** in **dodeljevanje datotek v fizičnem pomnilniku**.

6.3.3.1. Kopija ob zapisu (Copy-on-Write)

Stranjenje na zahtevo se uporabi, ko se izvršljiva (binarna) datoteka prebere iz diska in naloži v fizični pomnilnik.
Sistemski klic fork() naredi otroški proces kot duplikat starševskega procesa. Sistemski klic fork() deluje tako, da naredi kopijo strani starševskega procesa. Tehnika kopija ob zapisu dovoli, da na novo rojeni otroški proces uporablja (si deli) enak naslovni prostor kot starševski proces. Ta naslovni prostor ali strani imajo oznako "**kopija ob zapisu**", kar pomeni, da če hoče proces pisati v skupno stran se prej ustvari kopija te strani. Proces spremeni kopijo strani ne pa originala. Torej strani se kopirajo samo, če proces želi spreminjati stran v skupni rabi. Vse nespremenjene strani pa si delijo starševski in otroški procesi in s tem ne zasedajo odveč pomnilnika.
Tako tehniko uporabljajo operacijski sistemi kot so Windows 2000, Linux in Solaris 2.

6.3.3.2. Dodeljevanje datotek v fizičnem pomnilniku (Memory-Maped Files)

Zamislimo si zaporedno branje datotek s standardnimi sistemskimi klici open(), read(), write(). Vsakič, ko dostopamo do datoteke je potreben sistemski klic in dostop do diska. Tehnika omogoča, da se del logičnega naslovnega prostora porabi za logično dodelitev datoteke namesto fizičnega pomnilnika. Blok na disku, ki predstavlja datoteko se dodeli strani v fizičnem pomnilniku. Začetek dostopa do datoteke se nadaljuje z uporabo strajenja na zahtevo, ko pride do napake strani. Prebere se del datoteke, ki je velik kot ena stran v fizičnem pomnilniku in se vanj prenese.
Taka tehnika torej omogoča, da zaporedno dostopanje do diska ne poteka preko sistemskih klicev, ampak kar preko običajnega dostop do fizičnega pomnilnika.

6.3.4. Vprašanja za ponavljanje

1. Kaj omogoča tehnika virtualnega pomnilnika?

2. Kakšne prednosti prinaša tehnika virtualnega pomnilnika?

3. Pod kakšnimi pogoji pride do napake strani? Kaj naredi operacijski sistem, ko pride do napake strani?

6.4. Vmesnik datotečnega sistema

Za večino uporabnikov je datotečni sistem najbolj viden del operacijskega sistema. Zagotavlja mehanizem za shranjevanje in dostop do podatkov in programov tako operacijskega sistema kot uporabnikov računalniškega sistema. Datotečni sistem sestavljata dva glavna dela:

- **zbirka datotek**, za shranjevanje podatkov in
- **struktura direktorijev** (map), ki poskrbi za organizacijo in informacijo o vseh datotekah v sistemu
- **razdelki** (particije), ki skrbijo za fizično ali logično ločitev strukture direktorijev

6.4.1. Koncept datoteke

Računalniki lahko shranjujejo informacije na različne shranjevalne medije (magnetne diske, magnetne trakove, optične diske, ...). Operacijski sistem omogoča enoten logičen pogled na shranjene informacije in se ne posveča fizičnim lastnostim medija. Enota za logično shranjevanje se imenuje **datoteka**. Datoteke so dodeljene shranjevalnim napravam preko operacijskega sistema. Shranjevalne naprave ohranijo informacije permanentno, tudi po izpadu električne energije ali ponovnem zagonu sistema.
Z uporabniškega pogleda je datoteka najmanjši del logičnega sekundarnega shranjevanja. Podatke ne moremo shraniti na sekundarni medij dokler jih ne ovijemo v tako imenovano datoteko.
V datoteko lahko shranimo različne tipe podatkov. Vsak tip datoteke ima značilno **strukturo**:

- **tekstovna datoteka** – zaporedje znakov organiziranih v vrstice ali strani
- **izvorna datoteka** (source file) – zaporedje podrutin in funkcij, od katerih je vsaka kasneje organizirana kot deklaracija z izvršljivimi stavki
- **objektna datoteka** – zaporedje bytov, ki so organizirani v bloke, ki jih razume sistemski povezovalec (linker)
- **izvršilna datoteka** – zaporedje kode razporejeno v sekcije, katero nalagalnik prenese v pomnilnik in začne izvajanje.

6.4.1.1. Lastnosti datoteke

Datoteka ima naslednje lastnosti (atribute):

- **identiteto** – unikatna lastnost, ki je običajno številka in identificira datoteko znotraj datotečnega sistema. Na identiteto so priklopljene še ostale lastnosti datoteke
- **ime –** edina informacija, ki je v človeku razumljivi obliki
- **tip** – informacija, ki pride v poštev sistemom, ki podpirajo različne tipe datotek.
- **lokacija** – informacija je kazalec na napravo in na lokacijo datoteke na tej napravi
- **velikost** – predstavlja trenutno velikost datoteke izražene v enotah byte, blok
- **zaščita** – kontrola dostopa do datoteke, ki pove, kdo lahko iz datoteke bere, datoteko izvaja in v datoteko piše
- **čas, datum in lastništvo** – informacija lahko pove kdaj je datoteka nastala, je bila nazadnje spremenjena, nazadnje uporabljena, kdo je datoteko ustvaril, ...

Informacije o vseh datotekah so shranjene v direktoriju v katerem so datoteke shranjene. Tipično je vsebina direktorija definirana z **imenom datoteke** in **njeno identiteto**.

6.4.1.2. Operacije nad datoteko

Operacijski sistem priskrbi sistemske klice, ki ustvarijo, pišejo v datoteko, berejo iz datoteke, prestavijo datoteko, izbrišejo datoteko in brišejo vsebino datoteke.

- **ustvarjanje datoteke** – najprej se mora najti prostor v sistemu, nato se mora narediti zapis v direktoriju, ki pove kako je datoteki ime in kje v datotečnem sistemu se nahaja
- **pisanje v datoteko** – pisanje povzroči sistemski klic, kjer moramo določiti ime datoteke in informacije, ki naj se zapišejo v datoteko. Ko podamo ime datoteke sistem poišče v direktorijih datoteko in njeno lokacijo v datotečnem sistemu
- **branje iz datoteke** – sistemskemu klicu za branje datoteke moramo podati ime in blok v pomnilniku, kjer bo začasno prebrana datoteka
- **premestitev znotraj datoteke** – poiščemo zapis datoteke v direktoriju in trenutna pozicija datoteke se nastavi na podano vrednost. Premestitev znotraj datoteke ne zahteva nobene V/I operacije.
- **brisanje datoteke** – poiščemo zapis v direktoriju in sprostimo ves prostor, ki ga zaseda datoteka. Prostor je tako na voljo drugim datotekam.
- **brisanje vsebine datoteke** – ko se vsebina datoteke pobriše, se vsi atributi ohranijo, razen velikosti.

Poleg teh osnovnih operacij poznamo še operacije nad datotekami, ki so sestavljenka osnovnih operacij (preimenovanje, dodajanje vsebino na konec datoteke (append), kopiranje, prenašanje, spreminjanje atributov datoteke.

Da se nebi iskanje datotek vseskozi ponavljalo operacijski sistem vzdržuje informacijo o odprtih datotekah – **tabela odprtih datotek**. Ko se zahteva operacija nad datoteko je datoteka iskana kar preko kazalca v tej tabeli, tako da iskanje ni potrebno. Nekateri sistemi posebej zahtevajo najprej sistemski klic "open" (poišče datoteko in njeno lokacijo zapiše v tabelo odprtih datotek) in nato šele operacijo nad datoteko.

6.4.1.3. Tipi datotek

Najprej se vprašamo ali operacijski sistem razlikuje različne tipe datotek. Kaj bi bilo, če bi sprožili ukaz za tiskanje na binarni datoteki. Tiskalnik bi tiskal nerazumljive znake. Če operacijski sistem pozna tip datoteke, bi omenjeno tiskanje preprečil.

Tip datoteke najlažje implementiramo tako, da poleg imena datoteke dodamo še končnico ponavadi ločeno s piko (MS-DOS uporablja ime datoteke dolgo do 8 znakov + pika . + končnica dolga 3 znake – npr. readme.txt).

Operacijski sistem uporablja končnico, da ve kakšen je tip datoteke in kakšne operacije so dovoljene nad datoteko.
Samo datoteke z končnico .exe, .com in .bat se lahko izvršujejo v operacijskem sistemu MS-DOS. Končnici .exe in .com sta datoteki v binarni izvršljivi obliki, medtem ko .bat tip datoteke vsebuje ASCII znake, ki so ukazi operacijskemu sistemu.
Seveda obstajajo tudi končnice, ki so povezane s programi (npr. prevajalnik pričakuje izvorno datoteko s končnico .asm, urejevalnik besedil Word pričakuje datoteko s končnico.doc, itd...).

file type	usual extension	function
executable	exe, com, bin or none	read to run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rrf, doc	various word-processor formats
library	lib, a, so, dll, mpeg, mov, rm	libraries of routines for programmers
print or view	arc, zip, tar	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, rm	binary file containing audio or A/V information

Različni operacijski sistemi uporabljajo različne označbe tipa datoteke:

- Macintosh uporablja za končnico "text" ali "pict",
- Unix ne podpira takega načina označevanja datotek, saj uporablja surovo **magično številko**, ki je shranjena na začetku nekaterih datotek in ta številka označuje tip datoteke. Čeprav tudi pri Unix-u opazimo končnice datotek, so te končnice le v pomoč uporabniku, ne pa operacijskemu sistemu. Končnice so stvar programerja ali jih v svojih programih uporablja ali ne.

6.4.1.4. Struktura datoteke

Spomnimo se, da ima magnetni disk dobro definirane dolžine blokov, ki so definirani z velikostjo sektorja. Vse V/I operacije diska vsebujejo enote bloka (fizični zapis). Vsi bloki so enakih velikosti (512Byte – 4096Byte). Logični zapis se seveda ne more vedno ujemati s fizičnim zapisom. Logični zapisi spreminjajo dolžino, zato je **pakiranje** nekega števila logičnih zapisov v fizične bloke najbolj običajna rešitev pri zapisu logičnih blokov v fizične bloke.
Unix definira vse datoteke kot zaporeden tok bytov (8 bitov). Vsak byte je individualno naslovljen z odmikom od začetka (ali konca) datoteke. V tem primeru je 1 logični zapis = 1 byte. Tako se 1 blok (sektor) na disku zapolni s 512 logičnimi zapisi.
Logični zapis, fizični zapis in tehnika pakiranja določajo koliko logičnih zapisov bo v fizičnem bloku. Pakiranje je lahko določeno s strani programerja (aplikacije) ali operacijskega sistema.
Ker je prostor na disku dodeljen preko blokov, je nekaj blokov, ki niso popolno zasedeni. Če so bloki veliki 512 bytov in je datoteka velika 1025 bytov potem datoteka zasede 3 bloke po 512 bytov. Natančno zasede 2 bloka in še 1 byte v 3 bloku. Koliko prostora je izgubljenega? Skoraj celoten blok (511 bytov)!!!. Če vse dodeljujemo v blokih namesto v bytih pridemo do izgubljenega prostora in problema, ki mu pravimo notranje drobljenje (internal fragmentation). Vsi datotečni sistemi trpijo za notranjo drobitvijo in je tem večja tem večji so bloki.

6.4.1.5. Metode dostopa

Datoteke shranjujejo informacije. Ko so te informacije potrebne moramo do njih dostopati in jih prenesti v pomnilnik. Informacija v datoteki je lahko dosežena na več načinov. Nekateri sistemi podpirajo samo en način dostopa do datotek, drugi podpirajo več in tako izbirajo v danem trenutku optimalnega.

6.4.1.5.1. Zaporeden dostop

To je najenostavnejši dostop, kjer se informacije iz datoteke prenašajo po vrsti – en zapis za drugimi. Večina operacij nad datoteko je branje in pisanje. Operacija branja prebere naslednji zapis v datoteki in avtomatsko poveča interni kazalec v datoteki, ki zasledi naslednjo V/I lokacijo.
Omenjen dostop bazira na dostopu do datotek na magnetnem traku in deluje dobro tako na sekvenčno dostopnih napravah (magnetni trak) kot na naključno dostopnih napravah (magnetni disk, optični disk).

6.4.1.5.2. Direktni dostop

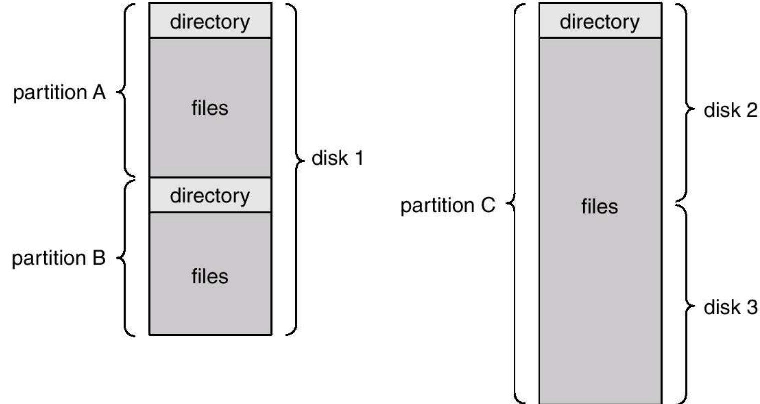
Povedali smo že, da je datoteka sestavljena iz logičnih zapisov fiksne velikosti, ki omogoča programom branje in pisanje zapisov. Metoda direktnega dostopa je mogoča na naključno dostopnih napravah. Datoteka je vidna kot oštevilčen zaporeden seznam blokov ali zapisov. Direktni dostop ali relativni dostop omogoča da beremo ali pišemo v točno določen blok ali zapis. Tako lahko preberemo blok 14 nato blok 53, itd.. Pri operacijah z metodo direktnega dostopa je potrebno definirati številko bloka kot parameter (read n, write n, kjer je n številka bloka).
Številka bloka, ki ga poda uporabnik operacijskemu sistemu je ponavadi **relativna številka bloka**. Relativna številka bloka je indeks, ki je relativen glede na začetek datoteke (prvi relativni blok v datoteki je 0, naslednji 1, itd.), četudi je fizična pozicija bloka 0 14703 in bloka 1 3192. Uporaba relativnih številk blokov omogoča operacijskemu sistemu odločitev kam bo datoteka postavljena (**problem razporeditve** bomo videli pozneje).

6.4.2. Struktura direktorija

Sistemi lahko shranijo na milijone datotek na magnetnih diskih visokih kapacitet. Da bi lahko obvladovali množico datotek jih moramo smiselno organizirati. Organizacija poteka v dveh korakih:

- disk je potrebno razdeliti na enega ali več razdelkov (particij - "minidiski" v svetu IBM in "volumes" v svetu PC in Macintosha). Vsak disk v sistemu vsebuje vsaj en razdelek, ki je nizko-nivojska struktura v kateri so direktoriji in datoteke. Nekje je razdelek del enega diska, drugje pa lahko razdelek sega čez več diskov. Razdelke lahko gledamo kot virtualne diske in tako lahko imamo na enem disku nameščenih več operacijskih sistemov (vsak sistem ima lahko svoj datotečni sistem).

- vsak razdelek vsebuje informacije o datotekah, ki so na razdelku. Te informacije so shranjene v **direktoriju naprave** ali **tabeli vsebine prostora**. Direktorij naprave ali enostavneje direktorij hrani informacije kot so ime, lokacija, velikost, tip vseh datotek, ki ležijo na razdelku (glej sliko).

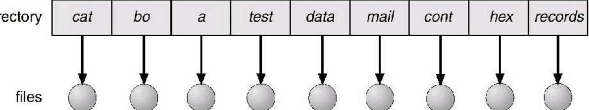


Direktorij lahko gledamo kot simbolno tabelo, kjer se imena datotek pretvarjajo v zapis v direktoriju.
Nad direktorijem morajo biti omogočene naslednje operacije:

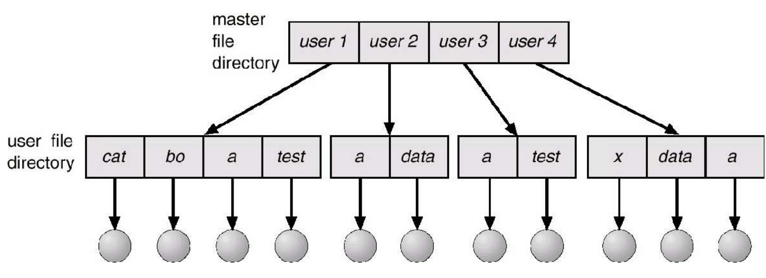
- **iskanje datotek** – najti želimo določeno datoteko znotraj direktorija (tudi preko določenega vzorca (*.exe))
- **ustvarjanje datoteke** - nova datoteka mora biti dodeljena prostoru na disku in njena lokacija, ime, ... zapisana v zapis (tabelo) v direktoriju
- **brisanje datoteke** - ko datoteke ne potrebujemo več se zapis datoteke v direktoriju pobriše (vsebina na disku pa ostane in je nedefinirana – prosta)
- **listanje vsebine direktorija** - želimo videti vsebino direktorija in lastnosti vseh datotek znotraj direktorija
- **preimenovati datoteko** - želimo dostopati do vsakega poddirektorija in vsake datoteke znotraj datotečnega sistema (backup datotečnega sistema)
- **prečkanje datotečnega sistema** – doseči hočemo vsak direktorij in datoteko znotraj strukture direktorija

6.4.2.1. Eno-nivojski direktorij

Vse datoteke so shranjene v istem direktoriju. Pomanjkljivost se pokaže, ko se nabere veliko datotek ali ko ima sistem več uporabnikov.



6.4.2.2. Dvo-nivojski direktorij



V prejšnjem nivoju strukture direktorija je bil problem ko sta dva uporabnika v istem direktoriju shranila datoteko z istim imenom. Problem na dvo-nivojskem direktoriju rešimo z direktorijem za vsakega uporabnika posebej. Tako ima vsak uporabnik sistema **uporabniški datotečni direktorij** (user file directory (UFD)). Ko se uporabniško opravilo zažene ali se uporabnik prijavi v sistem se išče po **glavnem datotečnem direktoriju** (master file directory (MFD)). MFD je indeksiran z uporabnikom ali številko uporabnika in vsak zapis kaže na UFD uporabnika. Ko uporabnik hoče datoteko se vse operacije izvajajo znotraj UFD.
S tem smo rešili problem kolizijo imen datotek. Sistem strogo ločuje uporabniške datoteke in če vsak posameznik deluje neodvisno je to v redu, če pa hočejo uporabniki sodelovati preko datotek je tale rešitev slaba.

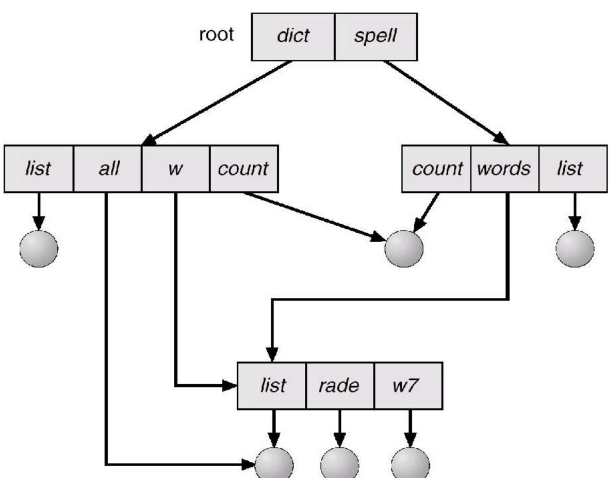
6.4.2.3. Drevesna struktura direktorija

Drevesno struktura verjetno najbolj poznamo saj se z njo srečamo vsakič, ko smo za računalnikom.
Po drevesni strukturi se sprehajamo preko poti. Imena poti so lahko:

- **absolutna**, kjer se ime začne z na najvišjem direktoriju in se nadaljuje po strukturi do datoteke (c:\Documents and settings\jakak\navodila.txt po Microsoftovo, /home/jakak/navodila.txt po Unixovo) in
- **relativna**, kjer se ime začne v direktoriju kjer smo trenutno postavljeni (..\metkas\navodila.txt po Microsoftovo, ../metkas/navodila.txt po Unixovo)

Če uporabniku pustimo, da ustvari direktorij mu s tem omogočimo, da si organizira svoje datoteke.
Poglejmo kako izbrišemo direktorij. Če je direktorij prazen ga enostavno izbrišemo, če pa ni moramo najprej izbrisati vsebino, ki je lahko datoteka ali poddirektorij in potem šele sam direktorij (Pri MS-DOS je bilo treba najprej izbrisati vsebino potem direktorij, pri Unix imamo ukaz rm, ki lahko izbriše vsebino in direktorij). Vendar pa je tak ukaz lahko zelo nevaren, saj lahko povzroči nepredvidljivo izgubo podatkov.

6.4.2.4. Acikličen diagram direktorijev



Zamislimo si dva programerja, ki delata na projektu, kjer je potrebno skupno delo. Datoteke, ki pripadajo projektu so shranjene v poddirektoriju, ločenem od uporabniške strukture uporabnikov. Oba uporabnika bi želela imeti projektni direktorij v svoji strukturi. Projektni direktorij bi torej morala dati v **skupno rabo**. Direktorij ali datoteka v skupni rabi obstaja istočasno na dveh ali več mestih v datotečnem sistemu. Drevesna struktura prepoveduje skupno rabo datotek ali direktorijev, kar aciklična ne. Aciklična struktura je naravna posplošitev drevesne strukture. Treba je ločiti kopijo datoteke in skupno rabo datoteke.

Če uporabljamo kopijo datoteke in en uporabnik spremeni kopijo, se sprememba ne bo odrazila v originalu datoteke. Tako drugi uporabnik ne bo mogel videti sprememb, ki jih je vnesel prvi uporabnik.

Če porabimo skupno datoteko se bo sprememba v datoteki ali direktoriju poznala tudi ko bo drugi uporabnik dostopal do datoteke ali direktorija.

Skupno rabo datoteke ali direktorija implementiramo z novim vnosom v direktoriju imenovanim **povezava** (link). Povezava je kazalec na neko datoteko ali poddirektorij. Povezavo lahko naredimo z absolutno ali relativno potjo. Povezava je posebno označena v zapisu direktorija, tako da se loči od ostalih datotek in poddirektorijev. Operacijski sistem tako prezre povezave, ko prečka strukture direktorijev in s tem ohranja aciklično strukturo sistema.

Aciklična struktura je bolj fleksibilna od drevesne strukture, vendar je bolj kompleksna. Poglejmo nekaj problemov, ki se lahko pojavijo.

Datoteka ima sedaj lahko veliko absolutnih imen poti. Brisanje skupne datoteke pusti za sabo mrtve povezave ali kar je še huje, na mestu skupne datoteke se sedaj pojavi neka druga datoteka in kazalec na fizični blok sedaj kaže nekam na sredino nove datoteke.

Reševanje težav z brisanjem skupne datoteke je lahko rešeno na dva načina.

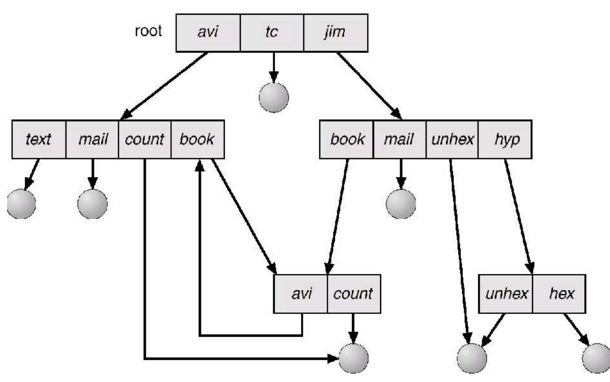
Prvi način uveljavlja **simbolične povezave** (simbolic link), kjer brisanje simbolične povezave ne vpliva na ciljno datoteko – skupna datoteka ne ve, da nanjo kaže kakšna povezava. Če pobrišemo skupno datoteko, ki ima simbolične povezave, se povezave ohranijo in delo brisanja sedaj slepih povezav pusti uporabniku. Uporabnik se tega zaveda šele ko uporabi povezavo in dobi napako napačno ime datoteke (invalid file name).

Tako Unix kot Microsoft uporabljata simbolične povezave.

Drugi način prepoveduje brisanje skupne datoteke, dokler ni zadnja referenca nanjo pobrisana. Datoteka vzdržuje **referenčni list**, kjer ko ustvarimo povezavo vpiše nastalo povezavo in ko povezavo izbrišemo se vpis izbriše. Torej skupno datoteko lahko izbrišemo, ko je referenčni list prazen. Problem se tu pojavi kako velik je referenčni list. Ni nam treba vpisovati, kje je povezava ampak nam zadostuje že številka, ki pove koliko povezav je narejenih. Vpis povezave poveča števec, brisanje povezave zmanjša števec. Datoteka nima več povezav, če je njen števec v referenčnem listu enak 0. Povezava, ki vpliva na ciljno datoteko s tem ko spreminja referenčni list pravimo **trda povezava** (hard link) in se

uporabljajo v Unixu.
Takih problemov nima MS-DOS, ki onemogoča skupne rabe direktorijev in datotek, zato pravimo, da ima MS-DOS operacijski sistem drevesno strukturo.

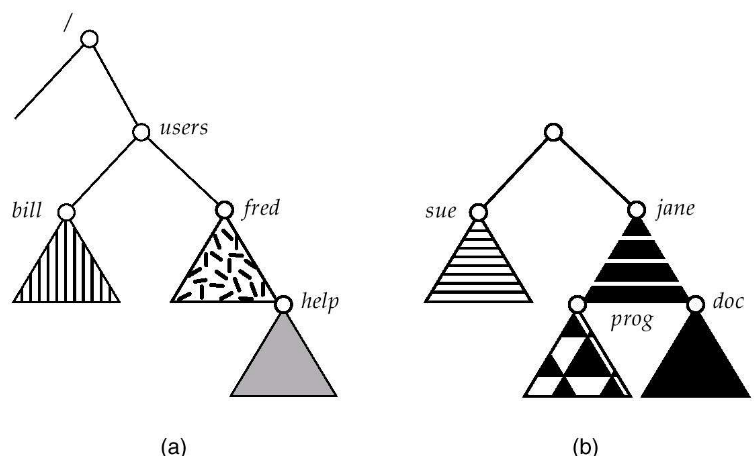
6.4.2.5. Splošen diagram direktorijev



Problem acikličnega diagrama je prav naloga, da ne dopušča ciklanja. Če začnemo z dvo-nivojsko strukturo in dopustimo ustvarjanje poddirektorijev pridemo do drevesne strukture. Ko damo v drevesno strukturo povezave poderemo drevesno strukturo in pridemo do diagrama enostavne strukture. Prednost acikličnega diagrama je relativna enostavnost za algoritme, da ugotovijo ali je na datoteki še kakšna referenca. Hočemo se izogniti prečkanju skupnega prostora dvakrat (odzivni čas) – če smo iskali datoteko v nekem velikem poddirektoriju in je nismo našli, se želimo naslednjič izogniti iskanju istega poddirektorija.

6.4.3. Priklop datotečnega sistema

Tako kot mora biti datoteka odprta preden jo uporabimo, tako mora biti datotečni sistem priklopljen predenj lahko procesi dostopajo do njega.
Procedura priklopa zahteva, da operacijskemu sistemu podamo ime naprave, ki jo želimo priklopiti in mesto znotraj datotečne strukture, kjer bo datotečni sistem priklopljen ali **točka priklopa**. Točka priklopa je ponavadi prazen direktorij v katerem bo datotečni sistem priklopljen (npr. v Unixu je na obstoječi uporabniški direktorij /home priklopljen datotečni sistem uporabnikov ali na obstoječi direktorij /mnt priklopljen recimo Windows razdelek (fat32) in pripadajoča struktura, kjer zna Linux pisati in brati). Na sliki vidimo obstoječ korenski (root) direktorij "/ "in poddirektorij "/users", v katerem je priklopljen uporabniški datotečni sistem narisan nepriklopljen (b) recimo na sekundarnem disku na tretjem razdelku, kjer sta dva poddirektorija sue in jane. Ko hočemo priti v direktorij uporabnika sue podamo ukazu absolutno pot / users/sue. Operacijski sistem preveri ali ima naprava, ki se hoče priklopiti ustrezen datotečni sistem (linux operacijski sistem pozna **fat32** datotečni sistem (pisanje in branje), **ntfs** datotečni sistem (samo branje zaenkrat)).



Microsoft Windows 95, 98, NT in 2000 vzdržujejo razširjeno dvo-nivojsko strukturo direktorijev, kjer so naprave in razdelki predstavljeni s črkami. Razdelki imajo splošen diagram direktorijev v povezavi s črkami pogona. Operacijski sistem takoj poišče vse

naprave v sistemu in datotečne sisteme priklopi ob zagonu.
Pri Unixu je priklop naprav lahko ekspliciten in bolj kontroliran, kajti obstaja sistemska konfiguracijska datoteka, ki vsebuje spisek naprav in priklopnih točk za avtomatski priklop ob zagonu. Ostale naprave, ki niso navedene v konfiguracijski datoteki moramo priklopiti ročno.

6.4.4. Skupna raba datotek

Povedali smo že, daje uporaba skupnih datotek primerna pri timskem delu. Tako morajo uporabniško orientirani operacijski sistemi nuditi možnost dajanja datotek v skupno rabo.

6.4.4.1. Več uporabnikov

Ko operacijski sistem sprejme več uporabnikov pride do izraza skupna raba, imenovanje datotek in zaščita datotek. V tem primeru mora datoteka ali direktorij vsebovati tudi atribute o zaščiti, skupni rabi, ..., kateri pri eno uporabniškem sistemu niso potrebni.
Kot implementacijo skupne rabe in zaščite so uvedli pojem **uporabnika** (user), **skupine** (group) in **ostale uporabnike** (others). Uporabnik, kot lastnik datoteke ali direktorija lahko počne vse operacije, medtem ko skupina lahko počne omejene operacije in ostali uporabniki tudi lahko še bolj omejene operacije nad datoteko ali direktorijem.
Pri Unixu dobi vsak uporabnik **številko uporabnika** imenovano "user ID" ali UID. Pri Micorsoftu NT pa je to "security ID" ali "SID". Številke uporabnikov so unikatne. Enako velaj za skupine, kjer vsaka skupina ima svojo številko skupine "group ID" ali "GID". Atribut o lastniku in skupini je zapisan zraven ostalih atributov datotek ali direktorijev. Ko uporabnik hoče izvršiti neko operacijo (pognal je proces, katerega je sam lastnik) nad datoteko se številka uporabnika in atribut v datoteki primerjata in podata ustrezno informacijo procesu. Operacija je omogočena ali ni omogočena odvisno od varnostne politike na datoteki ali direktoriju.

6.4.4.2. Oddaljeni datotečni sistemi

Nastop računalniških mrež je omogočil komunikacijo med oddaljenimi računalniki. Računalniki povezani v računalniško mrežo omogočajo dajanje sredstev v skupno rabo tudi preko omrežja gledano znotraj enega podjetja kot tudi širše preko interneta. Skupna raba podatkov deluje v obliki datotek. Prva implementacija skupne rabe datotek je storitev FTP (file transfer protocol), kjer uporabniki ročno prenašajo datoteke preko ustrezne aplikacije, ki bazira na **FTP** protokolu (CuteFTP, ...).
Druga metoda je metoda **porazdeljenih sistemov** (distributed file systems – DFS), kjer so oddaljeni direktoriji vidni preko lokalnega računalnika.
Tretja metoda je podobna ftp in implementira reverzni ftp in to je **WWW** (Wirld Wide Web). Za dosego datotek na WWW je potreben internetni brskalnik (Internet explorer, Mozilla, Mozilla Firefox, ...).
Medtem ko ftp lahko deluje v **anonimnem** in **avtentikacijskem** načinu, je pri www anonimen način skoraj obvezen. DFS pa je tukaj zelo dosleden in je zato bolj kompleksen.
Anonimen način pomeni, da lahko tudi uporabnik, ki je na oddaljenem sistemu tuj (ni na listi uporabnikov sistema) prebere datoteke in jih lahko prenese na svoj sistem.
Avtentikacijski način pomeni, da do datotek lahko dostopajo samo tisti, ki so na listi uporabnikov sistema in se predenj dostopajo do datotek identificirati z uporabniškim imenom in geslom, ki se nato primerja s tistim na sistemu.

6.4.4.2.1. Model odjemalec – strežnik

Oddaljeni datotečni sistemi dovolijo drugemu sistemu, da si priklopi oddaljeni datotečni sistem na svoj lokalni datotečni sistem. V tem primeru sistem, ki ima datoteke je strežnik, sistem, ki datoteke želi pa odjemalec. Strežnik lahko streže več odjemalcem in odjemalec lahko priklopi datotečni sistem več strežnikov. Priklop je na nivoju naprave ali poddirektorija.
Identifikacija odjemalca zna biti težava. Odjemalec je lahko določen preko imena v omrežju ali ip naslova, ki je lahko **prevaran** ali imitiran od nepridiprava. Bolj zanesljiva je komunikacija preko kodiranih ključev, vendar mora biti zagotovljena kompatibilnost protokola za dekodiranje in kodiranje ključev.
Unix avtentikacija deluje preko odjemalca, tako, da se UID številka odjemalca preveri z UID številko na strežniku. Če sta enaki je dostop ali priklop mogoč sicer pa ne.

6.4.4.2.2. Porazdeljeni informacijski sistemi

Da se poenostavi upravljanje servisov na relaciji odjemalec-strežnik se uporablja **porazdeljen informacijski sistem** ali **porazdeljen imenski sistem**.
Domenski imenski sistem (DNS) priskrbi transformacijo imena računalnika v številčen naslov računalnika v omrežju (tudi Internetu).
Drugi porazdeljeni informacijski sistemi uporabljajo parametre kot so uporabniško ime/geslo/ID uporabnika/ID skupine. Tako je Sun Microsystems predstavil **mrežni informacijski servis** (network information service (NIS). Na enem mestu je shranjeval informacije o uporabnikih in računalnikih v nekem informacijskem sistemu. Slabost tega servisa je bila to, da je ob avtentikaciji uporabnika geslo potovalo po omrežju nekodirano (v čisti tekstovni obliki), kar pomeni varnostni problem. Kasneje se je servis NIS izboljšal v NIS+, ki pa je precej bolj kompleksen za implementacijo.
Microsoftova omrežja uporabljajo metodo CIFS, kjer se avtentikacija ali **mrežna prijava** izvede na podlagi uporabniškega imena/gesla. Če se uporabnički imeni in geslo na odjemalcu in strežniku ujemata je prijava sprejeta in odjemalec, lahko uporablja mrežna sredstva na strežniku. Microsoft uporablja dve porazdeljeni imenski strukturi. To je **prijava v domeno**. Novejša tehnologija pa je **aktivni direktorij** (active directory). Ko postavimo imenski sitem se uporabniki lahko avtenticirajo na vsakem odjemalcu in strežniku v postavljenem informacijskem sistemu.
Najnovejša tehnologija na področju porazdeljenih informacijskih sistemov pa je takoimenovani **LDAP** (lightweight directory-access protocol). LDAP je varen porazdeljen imenski mehanizem. Aktivni directory bazira na LDAP sistemu. V LDAP direktoriju se tako shranjujejo vse informacije o uporabnikih in virih sredstev na vseh računalnikih v nekem podjetju. Rezultat je **varna enkratna prijava** za uporabnike, ki bodo svojo avtentikacijo izkazali enkrat in nato imeli dostop do vseh računalnikov znotraj informacijskega sistema podjetja. Poenostavljena je sistemska administracija, ki se v tem primeru vodi na enem mestu.

6.4.5. Zaščita

Informacijo, shranjeno v računalniku moramo varovati pred:
- **fizičnim uničenjem** – lahko se zgodi, da zaradi problemov strojne opreme pride do izgube podatkov (napake v branju in pisanju na medij, izpadi elektrike, stik glave s površino diska, nesnaga, ekstremne temperature, ...). V tem primeru govorimo o

varnosti in zanesljivosti sistema. Podvajanje informacij in tehnike varnostnega kopiranja vodijo k boljši zanesljivosti sistema.
- **zlorabo** – drug uporabnik povzroči operacijo nad našo datoteko, ki ni zaželena (nalašč izbriše datoteko). V tem primeru govorimo o **zaščiti sistema**.

6.4.5.1. *Tipi dostopa*

Potreba po zaščiti datotek pomeni zmožnost dostopati do datotek. Imamo sisteme, ki ne dovolijo dostopa do datotek drugega uporabnika (ti ne potrebujejo posebej zaščite na nivoju datoteke ampak na nivoju operacijskega sistema) in sisteme, ki nimajo nikakršne omejitve dostopa (Win95, Win98, WinME). Oba omenjena tipa sistemov sta preveč ekstremna vsak v svojo smer. Optimalen je **kontroliran dostop** do datotek.
Zaščitni mehanizmi uporabljajo kontroliran dostop z omejevanjem tipa dostopa do datoteke. Tipe dostopa do datoteke lahko razdelimo v:
- **Branje** (read – r) – branje iz datoteke, kar pomeni tudi kopiranje datoteke
- **Pisanje** (write – w) – pisanje v datoteko, kar pomeni tudi preimenovanje datoteke
- **Izvajanje** (execute – x) – prenos datoteke v pomnilnik in njeno izvajanje
- **Dodajanje** (append) – zapis nove vsebine na konec datoteke
- **Brisanje** (delete) – brisanje datoteke in sprostitev prostora za nadaljnjo uporabo
- **Listanje** (list) – listanje imena in lastnosti datoteke

6.4.5.2. *Kontrola dostopa*

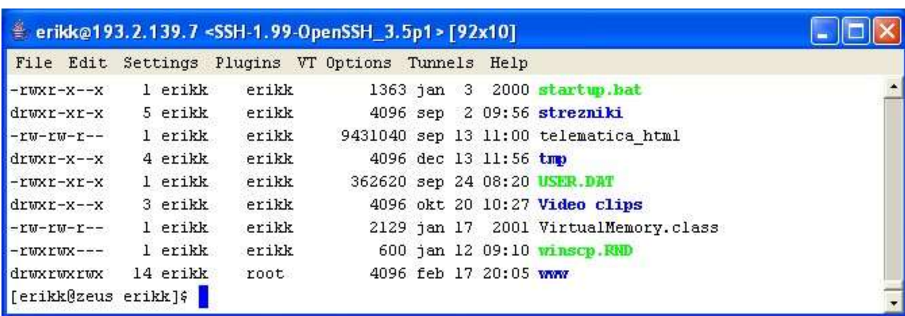
Najbolj običajna kontrola dostopa do datoteke je dodelitev datoteki ali direktoriju **kontrolni list dostopa** (access-control list – ACL), ki določa lastnika in tip dostopa vsakemu uporabniku.
Ko uporabnik dostopa do datoteke operacijski sistem primerja uporabnika in kontrolni list, ter na podlagi zapisa v kontrolnem listu uporabniku dovoli ali zavrne dostop (nastane zloraba zaščite in uporabniškemu procesu je onemogočeno dostopanje do datoteke).
Problem, ki se lahko pojavi je, da če želimo omogočiti dostop več uporabnikom se kontrolni list veča (novi uproabniki) in s tem nimamo več fiksnega zapisa v direktoriju ampak spremenljivega, kar vodi v zahtevnejše upravljanje s prostorom.
Problem rešimo s **stisnjenim kontrolnim listom** dostopa. Stisnjeni kontrolni list skrajša zapis v direktoriju, tako, da klasificira uporabnike v tri skupine:
- **lastnik** (user – u) – uporabnik, ki je datoteko ustvaril
- **skupina** (group – g)– niz uporabnikov, ki si delijo datoteko in potrebujejo podoben dostop
- **ostali** (other – o)– uporabniki, ki so registrirani na sistemu in niso latniki niti člani skupine

Kot primer si poglejmo pisanje knjige, kjer si pisatelj Simon, kot lastnik datoteke najame še dva korektorja Miha in Janez, ki pregledujeta slovnično pravilnost (damo jih v skupino "korektorji") in čim več kritikov njegove knjige.
Simon lahko na datoteki počne vse operacije, korektorja lahko bereta in pišeta, ne smeta pa pobrisati datoteke, kritiki pa lahko samo berejo datoteko.

V primeru Linux je zaščita definirana z 9 biti (3*3). Prvi trije biti predstavljajo tip dostopa za lastnika, drugi trije biti tip dostopa za skupino in zadnji trije biti tip dostopa za ostale uporabnike.

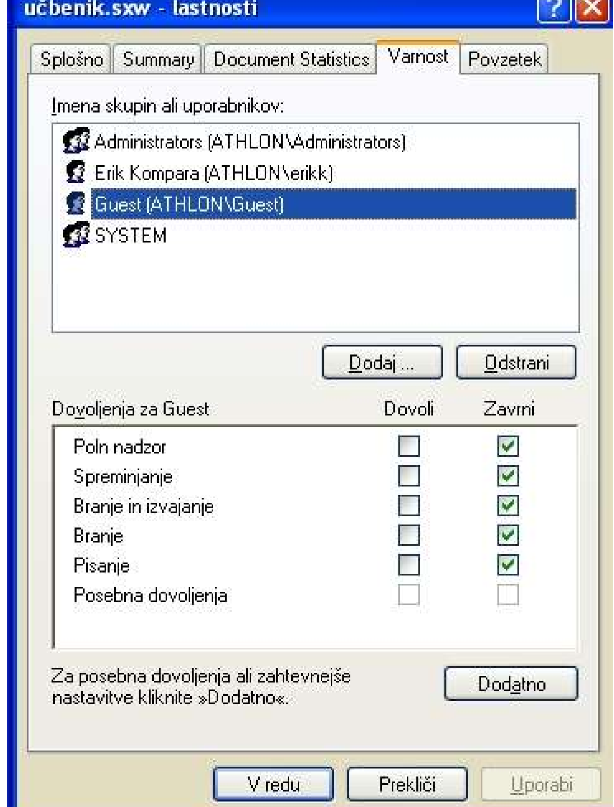
Torej datoteka knjiga bi imela v kontrolnem listu zapis: rwx rw- r--, bere, piše in izvaja lahko lastnik, bere in piše skupina korektorji in samo berejo pa vsi ostali uporabniki, ki so prijavljeni v sistemu.

Primer listanja vsebine direktorija v Linux sistemu, kjer vidimo vse lastnosti datoteke (zaščita, lastnik, skupina, velikost v bytih, mesec, dan, ura, minuta nastanka in ime datoteke ali direktorija):



V primeru Windows XP je najprej potrebno povedati, da kontrolo dostopa omogoča NTFS datotečni sistem (NT, 2000, XP).

Na sliki vidimo, da kako se uporabniku Guest onemogoči kakršenkoli dostop do datoteke učbenik.sxw.



6.4.6. Vprašanja za ponavljanje

1. Naštej operacije nad direktoriji.

2. Kaj je relativna in kaj absolutna pot?

3. Katera drevesna struktura omogoča skupno rabo datotek in direktorijev? Opiši razliko med uporabo kopije datoteke in skupne rabe datoteke na primeru dveh uporabnikov, ki sodelujeta.

4. Naštej porazdeljene imenske sheme in kaj je prednost porazdeljenega imenskega sistema?

5. Kaj je potrebno podati operacijskemu sistemu, da lahko priklopi datotečni sistem?

6. Kako lahko sistemi implementirajo kontrolo dostopa do datotek?

6.5. Implementacija datotečnega sistema

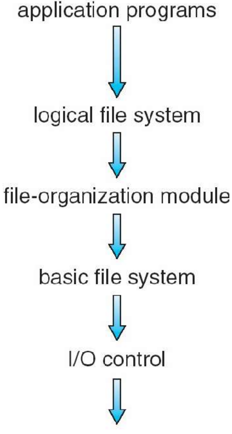
Poglavje se ukvarja z shranjevanjem in dostopom do datotek na sekundarnem mediju – magnetnem disku. Pogledali si bomo poti za uporabo datotek, dodeljevanje diskovnega prostora, sproščenega prostora, iskanje lokacij, kjer so shranjeni podatki, ...

6.5.1. Struktura datotečnega sistema

Zaradi učinkovitosti V/I sistema poteka prenos podatkov med diskom in pomnilnikom preko enot **blokov**. Vsak blok je eden ali več **sektorjev**. Odvisno od diska se velikost sektorjev razteza od 32 bytov do 4096 bytov. Običajno so veliki 512 bytov. Da zagotovimo učinkovit dostop do diska operacijski sistem naloži datotečni sistem, da omogoči podatkom shranjevanje, lociranje in dostopanje. Datotečni sistem zahteva dva različna problema dizajniranja:

- definirati kako bo datotečni sistem izgledal uporabniku (datoteka in njeni atributi, operacije nad datotekami, organizacija datotek v direktorije)
- definirati algoritme in podatkovne strukture, ki povezujejo logični datotečni sistem s fizičnim magnetnim diskom

Datotečni sistem je v splošnem sestavljen iz več nivojev.



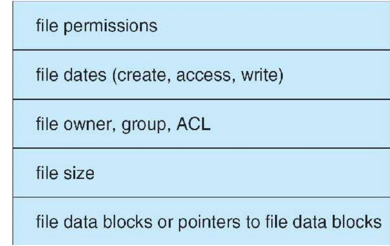
- Najnižji nivo **V/I kontrola** je sestavljena iz **gonilnikov naprave** in prekinitvenih rutin, ki prenese informacije med pomnilnikom in sistemom diska. **Gonilnik naprave** si lahko predstavljamo kot prevajalnik iz vhodnih višje-nivojskih ukazov (npr. retrieve block 126) v izhodne nižje-nivojske – strojne ukaze, ki so specifični za strojno opremo oziroma kontroler, ki deluje kot vmesnik med V/I napravo in ostalim sistemom.
- **Osnovni datotečni sistem** izvršuje generične ukaze gonilniku naprave, ki so branje in pisanje v fizične bloke na disku. Vsak fizični blok na disku je identificiran z njegovim numeričnim fizičnim naslovom (npr. drive 1, cylinder 73, track 2, sector 10).
- **Organizacijski modul datotek** poskrbi, da se prevede logični naslov bloka (npr. 0...N) v fizičen naslov, ki ga potrebuje osnovni datotečni sistem. Ta modul vsebuje tudi upravljavca prostega prostora na disku.
- **Logični datotečni sistem** vzdržuje strukturo datotečnega sistema (ne upravlja s podatki) in strukturo direktorijev preko kontrolnega bloka datoteke (FCB). Skrbi tudi za zaščito in varnost datotek in direktorijev.

Obstaja več implementacij datotečnih sistemov. Operacijski sistem lahko podpira več datotečnih sistemov. Na primer cd-mediji uporabljajo **High Sierra (ISO 9660)** format zapisa, ki je standard za izdelovalce CD-ROM naprav. Če standarda nebi bilo, bi operacijski sistemi imeli težave pri uporabi CD-ROM naprave. Na magnetnih diskih obstaja več datotečnih sistemov. Unix uporablja unix file sistem (UFS), kot osnova za ostale operacijske sisteme podobne Unixu (Linux uporablja ext2 in ext3 diskovni datotečni sistem, podpira pa tudi FAT in FAT32 za branje in pisanje in NTFS zaenkrat samo za branje). Microsoft NT podpira diskovne datotečne sisteme FAT, FAT32 in NTFS, ter tudi CD-ROM datotečne sisteme. V/I kontrolo in kodo osnovnega datotečnega sistema pa je domena vsakega operacijskega sistema posebej.

6.5.2. Implementacija datotečnega sistema

Implementacijo datotečnega sistema lahko razdelimo na strukture na disku in strukture v pomnilniku.
Struktura na disku lahko vsebuje informacije kako se operacijski sistem, ki je na disku zažene, število vseh in dodeljevanje prostih blokov na disku, strukturo direktorijev in datotek. Detajlno pa je na disku shranjen:

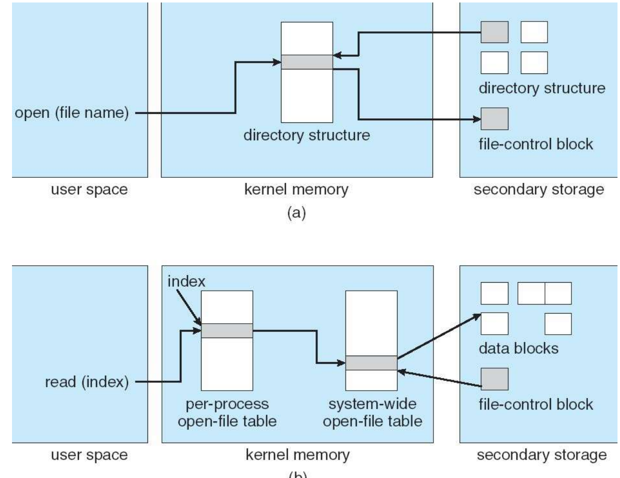
- **zagonski kontrolni blok** (boot control block) na razdelku, vsebuje informacije, ki jih potrebuje sitem, da zažene operacijski sistem iz razdelka na disku. Običajno je to prvi blok na razdelku.
  V UnixFS se imenuje *"boot block"*, v NTFS pa *"partition boot sector"*
- **kontrolni blok razdelka** (partition control block) na razdelku, vsebuje informacije o razdelku, kot so število blokov na razdelku, velikost bloka, število prostih blokov in kazalcev, število prostih kontrolnih blokov datotek in kontrolnih blokov kazalcev.
  V UnixFS se imenuje *"superblok"*, v NTFS pa *"Master File Table"*.
- **struktura direktorijev** za organizacijo datotek
- **kontrolni blok datoteke** (file control block - FCB) vsebuje več lastnosti datoteke (dovoljenja, lastništvo, velikost, lokacija podatkovnih blokov).
  V UnixFS se FCB imenuje *"inode"*, v NTFS je FCB shranjen znotraj *"Master File Table"*, ki uporablja relacijsko bazno strukturo – za vsako datoteko ena vrstica v bazi.



Strukture v pomnilniku so naslednje:

- **tabela razdelkov**, ki vsebuje informacije o vsakem priklopljenem razdelku
- **struktura direktorijev**, ki vzdržuje informacije o nazadnje obiskanih direktorijih.
- **tabela odprtih datotek na nivoju sistema**, ki vzdržuje kopijo FCB datotek, ki so odprte
- **tabela odprtih datotek za proces**, ki vsebuje kazalce v pripadajoči vnos v tabeli odprtih datotek na nivoju sistema

Če želimo ustvariti novo datoteko mora aplikacijski program poklicati logični datotečni sistem. Logični datotečni sistem pozna format strukture direktorijev. Da ustvari novo datoteko dodeli nov FCB, prenese direktorij v katerega želimo zapisati datoteko v pomnilnik, ga posodobi z novim imenom datoteke in FCB-jem, ter prenese iz pomnilnika nazaj na disk.



Unix rokuje z direktoriji enako kot z datoteko, le da je pri tipu datoteke oznaka, da gre za direktorij. Ostali operacijski sistemi tudi Windows NT uporabljajo različne sistemske klice za datoteke in direktorije.

6.5.2.1. Razdelki in priklapljanje razdelkov

Razdelek je lahko **surov** (raw) – ne vsebuje datotečnega sistema ali **pečen** (cooked) – vsebuje datotečni sistem. Surov razdelek uporabimo tam, kjer datotečni sistem ni primeren za uporabo (Unix uporablja izmenjalni prostor (swap space) na surovem razdelku, kjer uporablja svoj format na disku, nekatere baze podatkov in RAID sistemi potrebujejo surov razdelek).
Zagonska informacija je lahko shranjena na ločenem razdelku in ima svoj format, saj ob zagonu sistem nima naloženih gonilnikov naprave in ne more interpretirati datotečnega sistema. Zato je zagonska informacija v obliki zaporednih bytov, ki se naloži kot **zagonska slika** v pomnilnik. Izvajanje zagonske slike se začne na točno določeni lokaciji v pomnilniku. Tako lahko računalniki zaženejo več operacijskih sistemov, ki so na razdelkih na disku.
Zagonski nalagalnik (boot loader), ki pozna več datotečnih in operacijskih sistemov lahko zasede zagonski prostor in tako uporabniku ponudi izbiro, kateri operacijski sistem naj se zažene.
Korenski ali glavni razdelek, ki vsebuje jedro operacijskega sistema in potencialno še druge datoteke se mora priklopiti ob zagonu sistema. Ostali razdelki se lahko priklopijo avtomatsko ali ročno.
Microsoft Windows sistemi priklopijo razdelke v posameznih imenskih prostorih (C:, D:). V Unix operacijskih sistemih je priklop razdelka možen v vsakem direktoriju.

6.5.2.2. Virtualni datotečni sistem

Virtualni datotečni sistem omogoča, da se različni datotečni sistemi (različen datotečni sistem na drugem razdelku ali datotečni sistem na drugem računalniku) priklopi na lokalni datotečni sistem. Loči generične sistemske operacije od operacij, ki so značilne za implementacijo drugačnega datotečnega sistema. Loči se med lokalnimi in oddaljenimi datotekami.

6.5.3. Implementacija direktorija

Izbira algoritma za upravljanje in dodeljevanje direktorijev vpliva na učinkovitost datotečnega sistema.

6.5.3.1. Linearni seznam

Najenostavnejša metoda implementacije direktorijev je uporaba linearnega seznama imen datotek z kazalci na bloke podatkov. Linearni seznam zapisov direktorijev zahteva linearno iskanje določenega zapisa. Metoda je enostavna a časovno potratna. Ko želimo ustvariti datoteko moramo najprej preveriti, če datoteka s podanim imenom že obstaja. Nato dodamo nov vnos na konec direktorija.

6.5.3.2. Hash seznam

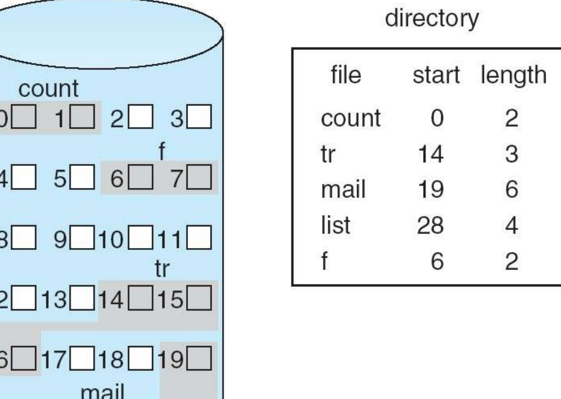
Pri hash seznamu se poleg linearnega seznama uporablja še hash podatkovna struktura. Hash seznam vsebuje številko vsake datoteke in kazalec na to datoteko v linearnem seznamu. Problem, ki nastane je spreminjanje velikosti hash seznama. Vzemimo direktorij, ki ima 64 vnosov datotek. Številka vsake datoteke je enaka ostanku pri deljenju zaporedne datoteke s 64 (2^6). Če v direktorij pride nova 65 datoteka moramo tabelo povečati na 128 (2^7) vnosov.

6.5.4. Metode dodeljevanja prostora

Direktni dostop do diska omogoča fleksibilno implementacijo datotek. Pogledali si bomo kako dodeliti prostor na disku datotekam na način, da bo prostor čimbolje izrabljen. Poznamo sosedno, povezano in indeksno dodeljevanje prostora.

6.5.4.1. Sosedno dodeljevanje prostora (contiguous allocation)

Vsaka datoteka zasede niz sosednjih blokov na disku. Naslovi diska definirajo linearno zaporedje na disku. Blok b+1 se zasede po bloku b. Tako dodeljevanje ne zahteva velikega premika glave.
Sosedno dodeljevanje prostora datoteki je definirano z naslovom diska in dolžino (v blokih) od prvega bloka. Če je datoteka dolga n blokov in se začne na lokaciji b, potem zasede bloke b, b+1, b+2, ..., b+n-1. Vnos v direktoriju za datoteko vsebuje naslov začetnega bloka in dolžino prostora, ki ga zaseda.

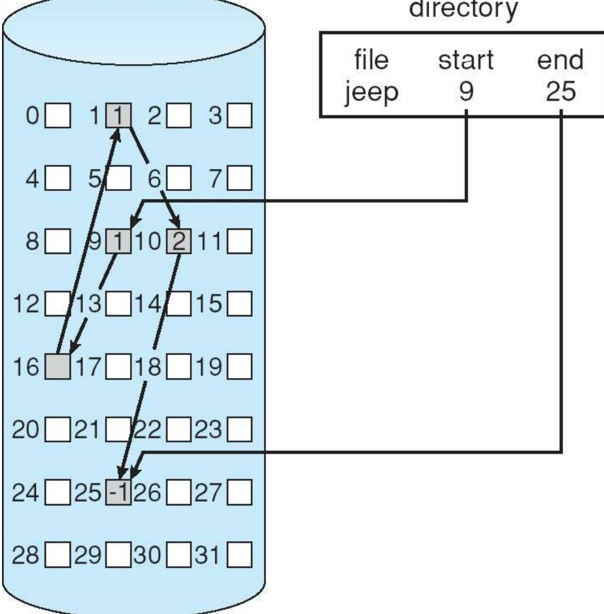


file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Dostop do datoteke, ki je bila dodeljena po tej metodi je zelo enostaven.
Pri **zaporednem dostopu** si datotečni sistem zapomni naslov diska zadnjega bloka, ki je bil uporabljen in uporabi naslednji blok.
Pri **direktnem dostopu** na blok i znotraj datoteke, ki se začne na bloku b lahko direktno naslovimo blok b+i.
Sosedno dodeljevanje podpira oba načina dostopa.
Prvi problem pri sosednjem dodeljevanju je iskanje prostora za novo datoteko. Ko govorimo o dinamičnem dodeljevanju pridemo do problema kako dodeliti n blokov dolgo datoteko v seznam lukenj (enako kot pri dodeljevanju prostora v pomnilniku). Spomnimo se metod dodeljevanja praznega prostora kot so prvo, najboljše in najslabše ujemanje. Tudi tukaj pride ob brisanju in pisanju v luknje do novih večjih ali manjših lukenj in problem se tudi tule imenuje **zunanje drobljenje**. Zunanje drobljenje obstaja, kadar je zaporedje blokov premajhno, da bi dodelilo prostor.
Drugi problem je ta, da se v naprej ne ve koliko bo datoteka velika, medtem ko ima že dodeljen prostor. Datoteki lahko dodamo podatke in ta postane večja. Vprašanje pa je ali je še prostor za dodane podatke v nadaljnjih sosednjih blokih. Zelo verjetno, da ne če smo uporabili metodo najboljšega ujemanja.

6.5.4.2. Povezano dodeljevanje prostora (linked allocation)

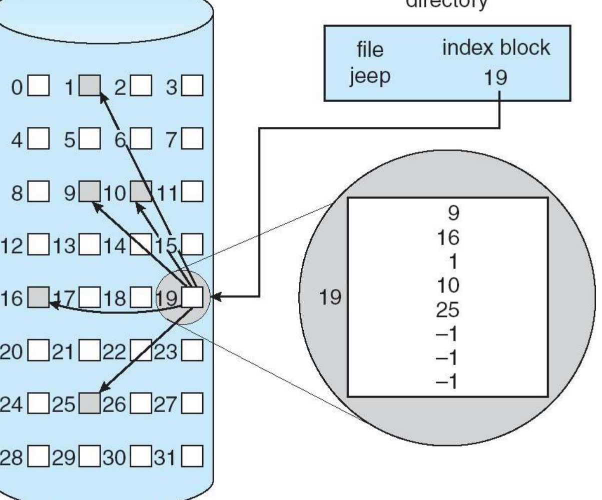
Povezano dodeljevanje reši vse probleme sosednjega dodeljevanja. Pri tem dodeljevanju je vsaka datoteka seznam povezav do blokov, ki sestavljajo datoteko. Datoteka je tako lahko razdrobljena po celem disku. Direktorij vsebuje kazalec na prvi in zadnji blok datoteke. Na primer datoteka dolga 5 blokov se začne na bloku 9, nadaljuje na bloku 16, potem na bloku 1, 10 in konča na bloku 25. Kazalci niso na razpolago uporabniku. Če je blok velik 512 Bytov in naslov diska (kazalec) porabi 4 byte potem uporabnik vidi bloke velike 508 Bytov.



Ustvarjanje nove datoteke povzroči vnos v direktorij, ki je sestavljen iz kazalca na začetni blok datoteke. Kazalec in velikost datoteke je 0, kar pomeni, da je datoteka prazna.
Pisanje v datoteko povzroči, da se preko sistema za upravljanje s prostim prostorom najde prosti blok, byti datoteke zapolnijo prosti blok in kazalec nanj se zapiše v zapis v direktoriju kot konec datoteke, itd....
Branje datoteke poteka tako, da kazalec v direktoriju pokaže na začetni blok datoteke, blok se prebere in ta kaže na naslednjega, ki se spet prebere, ...
Seveda ima ta metoda tudi slabosti.
Prva slabost je ta, da metoda omogoča efektivno uporabo zaporednega dostopa do datoteke. Direkten dostop je neefektiven. Za iskanje i-tega bloka v datoteki je potrebno vedno začeti iskanje na začetnem bloku datoteke in s kazalci na naslednji blok priti do i-tega bloka. Vsak dostop do kazalca pa zahteva branje z diska.
Druga slabost je koriščenje prostora na disku za kazalce. Če porabimo za kazalce 4 Byte od 512 Bytov, kolikor je blok velik smo porabili za kazalce 0,78% bloka. To pomeni, da bi disk z velikostjo 80 GB porabil 625MB za kazalce na bloke.
Problem se rešuje z zbiranjem več blokov v skupino (cluster). Običajno je 1 skupina = 4 bloke. To pripelje do tega, da je sedaj enota prenosa skupina ne pa blok. Seveda povečanje enote prenosa pripelje do več skupin, ki niso izkoriščeni do konca in s tem do notranjega drobljenja (internal fragmentation). Zamislimo si grozd velik 4 bloke po 512 B = 2048 B. Kaj se zgodi, ko shranimo datoteko veliko 1024 B. Polovica grozda je nezasedena! In kaj če imamo veliko majhnih datotek?
Tretja slabost je zanesljivost. Če je datoteka povezana s kazalci v blokih, ki so razmetani po celotnem disku in se recimo en kazalec izgubi (slab blok). Kaj če se kazalec zaradi napake v programu spremeni in kaže na napačen prostor ali celo na sredino druge datoteke.
Pomembna različica povezane metode dodeljevanja prostora uporablja **file allocation table** (FAT), ki ga uporablja operacijski sistem MS-DOS in OS/2.

6.5.4.3. Indeksno dodeljevanje prostora

Rešuje problem zunanjega drobljenja in problem povečanja velikosti datoteke pri sosednjem dodeljevanju prostora. Tudi problem direktnega dostopa je rešen tako, da se raztreseni bloki indeksirajo in dobijo zaporedje na enem mestu – **indeksnem bloku**. Vsaka datoteka ima svoj indeksni blok, ki je polje naslovov diska.



i-ti blok v indeksnem bloku kaže na i-ti blok datoteke. Direktorij vsebuje ime datoteke in naslov indeksnega bloka datoteke.

Branje i-tega bloka datoteke zahteva branje i-tega kazalca v indeksnem bloku in ta nam pokaže pot do željnega i-tega bloka.

Ko datoteko ustvarimo se ustvari indeksni blok, ki ima vse kazalce na 0, kar pomeni prazno datoteko. Ko se dodeli prvi blok preko upravitelja prostega prostora prvi kazalec kaže na zaseden blok (9), drugi kazalec na drugi zaseden blok (16), itd..

Seveda indeksno dodeljevanje prostora ima tudi slabosti.

Prva slabost je poraba prostora. Vsaka datoteka svoj indeksni blok in v primeru datoteke, kjer vsebina zasede 2 bloka je izguba prostora 33% (1 indeksni blok in 2 bloka vsebine). Če je veliko takih datotek je izguba prostora zelo velika. Torej indeksni blok bi moral biti čim manjši, vendar po drugi strani dovolj velik, da lahko shrani kazalce v primeru velike datoteke, ki zasede veliko blokov.

Zato obstajajo določeni mehanizmi, ki uravnavajo velikost indeksnega bloka.

- **povezana shema**: indeksni blok je velik 1 blok diska. Za vzdrževanje velike datoteke se blok razširijo tako, da se poveže z še enim indeksnim blokom. Naprimer indeksni blok bi vseboval ime datoteke in kazalce na prvih 100 blokov diska. Naslednji byte v bloku je 0, če je datoteka dovolj majhna ali kazalec na naslednji indeksni blok za večje datoteke.
- **večnivojski indeks**: Povezava poteka po nivojih, tako da prvo-nivojski indeks vsebuje kazalce na drugo-nivojske indekse, ki kažejo na bloke datoteke. Za dostop do bloka operacijski sistem najprej uporabi prvo-nivojski blok, ki pokaže na drugo-nivojski blok in ta blok na bloke datoteke. Tak postopek lahko zgradimo do tretjega ali četrtega nivoja odvisno od maksimalne dolžine datoteke.

  Če imamo blok velik 4096 B lahko vanj shranimo 1024 kazalcev po 4 B – 1024 indeksnih blokov na drugem nivoju. To pomeni, da dvo-nivojska struktura omogoča 1024*1024 = 1.048.576 blokov datoteke in s tem lahko shranimo **maksimalno veliko datoteko**: 1.048.576 * 4096 B = 4294967296 B=4194304 KB=4096 MB=**4GB**.

- **kombinirana shema**: uporablja jo UnixFS, ki drži recimo prvih 15 kazalcev indeksnega bloka v kontrolnem bloku datoteke (inode). Prvih 12 kazalcev kaže na **direktne bloke**, ki kažejo na bloke vsebine datoteke.
  Če je datoteka dovolj majhna (12 blokov), ni potrebe po dodatnem indeksnem bloku. Če je torej velikost bloka 4KB, lahko direktno dostopamo do 48KB podatkov (4KB*12 kazalcev).
  Naslednji 3 kazalci kažejo na **indirektne bloke**, ki lahko kažejo na **enojni, dvojni ali trojni indirektni blok**.
  Prvi od treh kazalcev ali 13 kazalec kaže na enojni indirektni blok, ki je indeksni blok in vsebuje kazalce na podatkovne bloke. (4MB velika datoteka)
  Drugi kazalec od treh ali 14 kazalec kaže na dvojni indirektni blok, ki vsebuje kazalce na enojne indirektne bloke, ki vsebujejo kazalce na podatkovne bloke. (4GB velika datoteka)
  Tretji kazalec sledi tej logiki.

Če uporabljamo kazalce dolžine 4B=32b, lahko naslovimo največ 2^{32} B = 4GB. Veliko različic Unix-a – Solaris, IBM-jev AIX, Linux uporablja 8B=64b kazalec, ta pa lahko naslovi terabytne datoteke.

6.5.5. Upravljanje s prostim prostorom

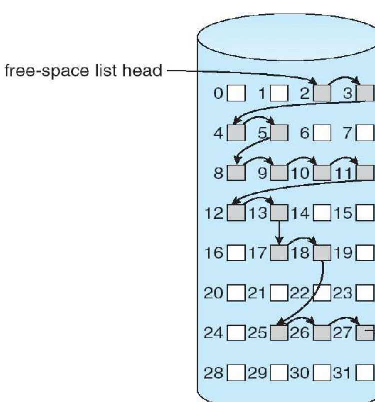
Ker je velikost diska omejena moramo sproščeni prostor (datoteka se izbriše) ponovno uporabiti. Za vzdrževanje prostega prostora skrbi **seznam prostega prostora**. Ta seznam vsebuje vse proste bloke na disku. Da ustvarimo datoteko najprej pogledamo v seznam prostega prostora za določen prostor in dodelimo proste bloke datoteki. Prostor se pobriše iz seznama prostega prostora. Ko je datoteka izbrisana, njen prostor na disku se doda na seznam.

6.5.5.1. Bit vektor

Pogosto je seznam prostega prostora implementiran kot **mapa z biti** ali **bit vektor**. Vsak blok na disku je predstavljen z 1 bitom. Če je blok zaseden ima bit vrednost 0, če je prost ima bit vrednost 1.

6.5.5.2. *Povezan seznam*

Ideja je v povezavi vseh prostih blokov na disku, kjer hranimo kazalec na prvi prosti blok na posebnem mestu diska in ga kopiramo v pomnilnik. Prvi prosti blok vsebuje kazalec na naslednji prosti blok itd.. Na sliki vidimo prvi blok (2), ki vsebuje kazalec na drugi prosti blok (3), ta na 4, itd..



6.5.6. Vprašanja za ponavljanje

1. Naštej iz katerih plasti je sestavljen datotečni sistem in kaj opravlja posamezna plast.

2. Kaj mora vsebovati magnetni disk, da lahko implementiramo datotečni sistem?

3. Ali mora vsak razdelek biti formatiran? Kakšen je še lahko razdelek?

4. Naštej nekaj datotečnih sistemov, ki jih uporabljajo različni operacijski sistemi.

5. Opiši sosednje dodeljevanje prostora in naštej slabosti metode.

6. Opiši povezano dodeljevanje prostora in naštej prednosti in slabosti metode.

7. Opiši indeksno dodeljevanje prostora in naštej prednosti, ter mehanizme dodeljevanja prostih blokov.

8. Koliko blokov lahko naslovimo s 2B kazalcem?

9. Koliko veliko datoteko lahko naslovimo po **indeksnem dodeljevanju z dvojnim indirektnim blokom**, če ima datotečni sistem bloke velike 2kB, pri tem uporablja kazalce velike 8B?
   Rešitev: 131072kB= 128MB

6.6. Strukture za masovno shranjevanje

Datotečni sistem lahko gledamo logično iz treh zornih kotov:
- uporabniški in programerski pogled na datotečni sistem
- pogled s strani operacijskega sistema
- najnižji nivo – pogled s strani strojne opreme sekundarnih (magnetni disk) in terciarnih (magnetni diski, CD, DVD mediji) naprav za masovno shranjevanje podatkov.

6.6.1. Struktura diska

Magnetni disk je glavni predstavnik sekundarnih pomnilnih naprav. Plošča, kot podatkovni nosilec meri od 1,8 do 5,25 cole in je prevlečena z magnetnim materialom. Razdeljena je na logične krožne **sledi**, ki so razdeljene na najmanjšo fizično enoto diska - blok ali **sektor**. V sektorje se zaporedno z magnetnim principom dodeljujejo logični bloki. Velikost logičnega bloka je ponavadi 512 B, vendar lahko na disku izvedemo nizkonivojski format in velikost blokov povečamo ali pomanjšamo. Skupina sledi, ki so vertikalno gledano na istem mestu oz. pod istimi r/w glavami imenujemo **cilinder**.
Sektor 0 je prvi sektor na prvi **sledi** (track), ki leži na najbolj zunanjem **cilindru**. Polnjenje blokov poteka zaporedno začenši na sektorju 0, nato do konca prve sledi, nato na ostalih sledeh istega cilindra in nato po ostalih cilindrih od zunaj proti noter.
Zaradi plošč, ki so v obliki kroga so na zunanji strani sledi daljše, na notranji strani kroga pa krajše. Če želimo da bo **pretok podatkov konstanten** lahko naredimo dvoje:
- ko se glava premika iz notranje sledi proti zunanji povečujemo hitrost vrtenja plošče in s tem ohranjamo **konstantno linearno hitrost** (costant linear velocity – CLV). Tako metodo uporabljajo CD in DVD enote.
- ko se glava premika po sledeh ne spreminjamo hitrosti vrtenja plošče in s tem ohranjamo konstantno kotno hitrost (constant angular velocity – CAV), vendar se mora **gostota bitov** na sledi zmanjševati s prehajanjem na zunanje sledi.

Kapaciteta diskov se meri v GB (Giga Byte). Bralno – pisalna glava leži tik nad površino plošče. Premika jo nosilec, ki je povezan z koračnim motorjem, katerega upravlja elektronika v tesni povezavi s kontrolerjem.
Ker je med glavo diska in površino mikro-meterska razdalja, obstaja velika nevarnost, da se glava dotakne površine diska. Kljub temu, da ima vrhnja plast zaščitni sloj lahko pride do poškodbe magnetnega sloja, čemur pravimo **stik glave s površino diska** (head crash). Poškodba je nepopravljiva!

6.6.2. Razvrščanje na disku

Ena od odgovornosti operacijskega sistema je učinkovita izkoriščenost strojne opreme.
Plošče diska se med delovanjem vrtijo s hitrostjo od 60 do 200 obratov/s. Hitrost dostopa do diska lahko razdelimo na dva dela.

- **prenosna hitrost** (transfer rate), koliko Bytov podatkov prenese disk od trenutka zahteve po V/I operaciji do trenutka zadnjega prenesenega Byta.
- **pozicijski čas** ali drugače **naključno-dostopni čas** (random-access time),
    - **dostopni čas** (seek time), čas v katerem se nosilec glave postavi na pravi cilinder
    - **čas rotacije**, (rotational latency), čas potreben, da pride sektor do položaja glave

Tipične prenosne hitrosti so nekaj MB/s in pozicijski časi pa nekaj ms.
S skrbno izbiro branja in pisanja na disk zmanjšamo dostopni čas in čas rotacije. Ponovimo, ko proces potrebuje V/I operacijo z ali v disk, proces izvede sistemski klic operacijskemu sistemu.
Zahteva poda naslednje informacije:

- ali je operacija pisanje ali branje
- naslov diska za prenos
- naslov fizičnega pomnilnika za prenos
- število Bytov, ki se bodo prenesli

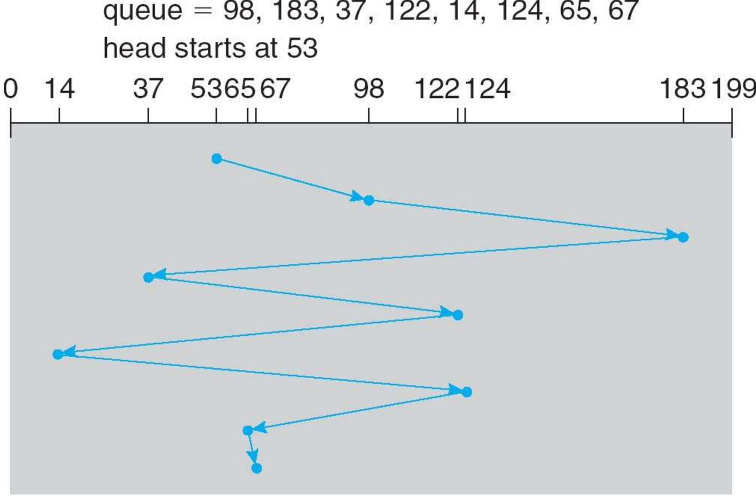
Če so zahtevani diski in kontroler na voljo, se zahteva nemudoma izvrši. Če je disk ali kontroler zaseden se se zahteva postavi v vrsto čakajočih na disk.

6.6.2.1. Prvi pride prvi melje(FCFS) razvrščanje

Najenostavnejša oblika razvrščanja na disku je FCFS algoritem. Je najbolj pravičen, a zato ni najbolj hiter. Zamislimo si čakalno vrsto diska, ki vsebuje zahteve za V/I operacije na na blokih naslednjih sledeh:

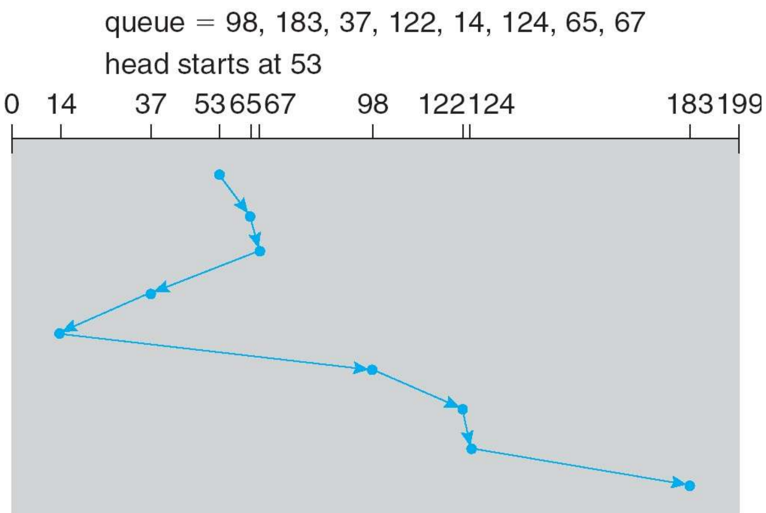
98,183,37,122,14,124,65,67

Če je glava diska na začetku na sledi 53, bo najprej serviral sled 98, potem 183, ... Skupno se bo glava premaknila za 640 sledi. Problem te metode je velik premik glave v primeru 122 sledi na 14 sled.



6.6.2.2. Najkrajši dostopni čas najprej (SSTF) razvrščanje

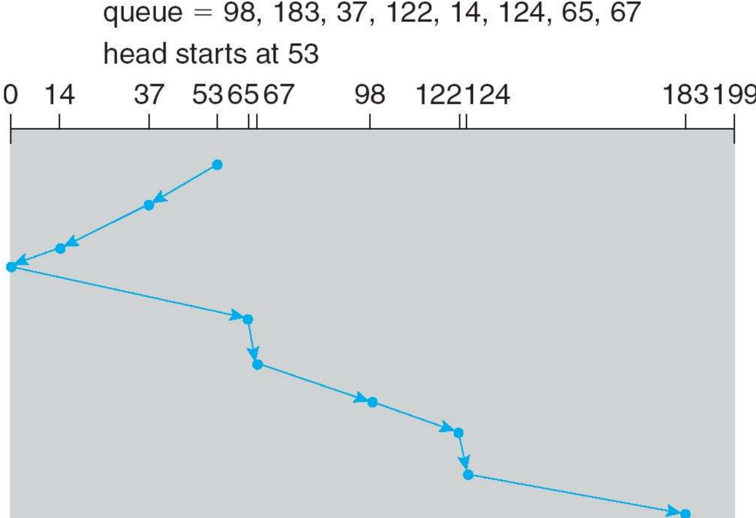
Zgleda razumljivo, da bomo najhitrejši, če bomo servirali zahteve, ki zahrevajo čimmanjši premik glave. To idejo uporablja algoritem najkrajšega dostopnega časa (shortest-seek-time-first SSTF). Metoda servira zahtevi, ki je najbližje trenutni postavitvi glave.
V našem primeru je sledi 53 najbliže sled 65, zato se glava premakne najprej tja. Ko smo na sledi 65 je najbližja sled 67, ... Tak način serviranja premakne glavo za skupno 236 sledi.



Problem takega algoritma, ki je podoben algoritmu pri razvrščanju procesov (shortest job first – SJF) je možnost **stradanja** nekaterih zahtev. Ker zahteve neprestano prihajajo si zamislimo sled 14 in 186. Serviranje zahteve na sledi 14 bo sledilo serviranje zahteve sledi 37, nato pride recimo pride nova zahteva za sled 24 in seveda servira njo. V teoriji do sledi 186 sploh nebi prišli ali pa bi prišli zelo pozno.
SSTF je boljši od FCFS vendar ne optimalen.

6.6.2.3. SCAN razvrščanje

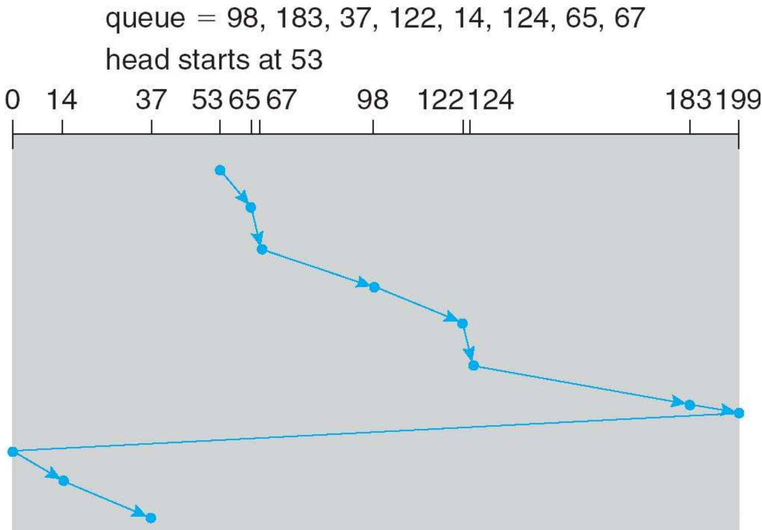
Glava diska potuje od enega konca do drugega in servira zahteve na katere naleti na tej poti. Ko je glava na koncu se potovanje in serviranje obrne v drugo smer.
Odvisno od tega v katero stran se glava premika v tisti smeri se bo serviranje začelo. Najprej 37, 14, konec strani, začetek potovanja v drugo smer, 65, 67, ...



Kaj se lahko zgodi pri tem razvrščanju? Če pride zahteva po sledi, ki je tik pred glavo, bo takoj servirana, če pa pride zahtev tik za glavo, bo pa servirana precej kasneje. Temu algoritmu pravimo tudi **dvigalo**, ker servira zahteve, kot dvigalo v hotelu servira ljudem.

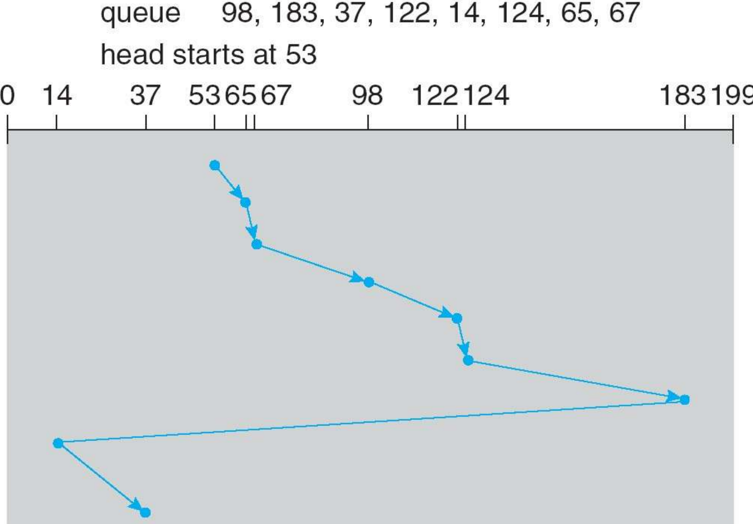
6.6.2.4. Krožni SCAN razvrščanje

Krožni SCAN ali C-SCAN algoritem pogleda v kateri smeri je večja gostota zahtev in v tisti smeri bo začel potovati do konca v eno smer in nazaj do konca v drugo smer. V našem primeru je gostota zahtev večja v desni smeri, zato bo začel servirati v desno do konca in nato v levo do konca.



6.6.2.5. LOOK razvrščanje

Oba algoritma SCAN in C-SCAN premakneta glavo do zadnje sledu, četudi na zadnji sledi ni kaj servirati. V tem algoritmu gre glava samo do zadnje zahteve v eni smeri in takoj zamenja smer. LOOK algoritem je različica SCAN algoritma in enako velja, da je C-LOOK različica C-SCAN algoritma, ker pogleda ali je še kakšna zahteva pred trenutno zahtevo.



6.6.3. Upravljanje z diskom

6.6.3.1. Formatiranje diska

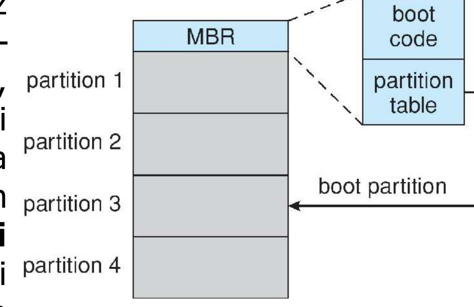
Nov magnetni disk je prazna plošča – magnetna plošča z snemalnim magnetnim materialom. Predenj na ploščo lahko shranimo podatke jo moramo razdeliti na sektorje, katere bo kontroler lahko bral in pisal. Proces se imenuje **nizkonivojsko formatiranje** ali **fizično formatiranje** (low-level formatting). Nizkonivojsko formatiranje zapolni disk z posebnimi podatki v v vsak sektor. Podatkovna struktura sektorja je sestavljena iz **glave**, **podatkovnega področja** (običajno 512 B) in **prikolice**. Glava in prikolica vsebujeta podatke, ki jih potrebuje kontroler diska (številka sektorja in koda za detekcijo napak (error-corecting-code – ECC)). Ko kontroler zapiše v sektor podatke se ECC posodobi in kaže sliko Bytov, ki so shranjeni v podatkovnem delu sektorja. Ko se sektor prebere se primerja slika v ECC in dejanska slika sektorja. Če obstaja razlika v slikah pomeni, da je sektor slab.

Da na disk shranimo datoteke, mora operacijski sistem zapisati svojo podatkovno strukturo na disk. To naredi v dveh korakih:

- **razdeljevanje diska** v razdelke (particije) – operacijski sistem razdeli enega ali več skupin sledi v razdelek. Operacijski sistem lahko obravnava razdelek kot posamezen disk. Zato je lahko na enem razdelku operacijski sistem (sistemske datoteke) na drugem razdelku uporabniške datoteke, itd....
- **logično formatiranje** – kreiranje datotečnega sistema. Operacijski sistem mora na disk zapisati osnovne podatke o datotečni strukturi (mape prostih in dodeljenih blokih (FAT ali inode) in začetni prazen direktorij)

6.6.3.2. Zagonski blok (boot blok)

Ko računalnik zaženemo ali ponovno zaženemo (reset) mora imeti sistem **začetni program**, ki naloži operacijski sistem. Začetni program je enostaven in inicializira vse dele sistema (registre procesorja, kontrolerje naprav, vsebino fizičnega pomnilnika) ter nazadnje zažene operacijski sistem. Začetni program poišče jedro operacijskega sistema na disku, ga naloži v fizični pomnilnik in skoči na lokacijo v pomnilniku kjer je začetek izvajanja operacijskega sistema in operacijski sistem se začne izvajati.



Začetni program je običajno zapisan v pomnilniku iz katerega se lahko samo bere (read-only-memory – ROM), poleg tega pa je fiksno in vedno na isti lokaciji, tako, da ko računalnik vžgemo procesor ve kje iskati prvi ukaz. Ker v ROM ne moremo pisati je ROM imun na računalniške viruse. Ponavadi je začetni program razdeljen na začetni program v ROM-u in **polni začetni program**, ki se nahaja na fiksni lokaciji na disku. Polni začetni program je shranjen v **zagonskih blokih** razdelka. Razdelek, ki vsebuje zagonske bloke je **zagonski disk** ali **sistemski disk**.

Koda v ROM pomnilniku pove kontrolerju diska, naj prenese zagonske bloke v fizični pomnilnik (sedaj še brez gonilnikov) in kodo začne izvajati. Polni začetni program je zmogljivejši od začetnega programa v ROM-u. Sposoben je pognati jedro operacijskega sistema iz nefiksne lokacije. Primer MS-DOS zagonskega diska prikazuje leva slika.

sector 0 | boot block
sector 1 | FAT
root directory
data blocks (subdirectories)

6.6.4. Vprašanja za ponavljanje

1. Naštej glavne dele magnetnega diska in opiši vrstni red dodeljevanja logičnih blokov v sektorje na disku.

2. Kako je sestavljen čas dostopa do bloka na disku?

3. Naštej načine razvrščanja na disku, opiši poljubni način in razloži razliko med ostalimi.

4. Kateri del zagona sistema naloži operacijski sistem in je tudi sposoben izbirati med različnimi operacijskimi sistemi?

7. Vhodno / Izhodni sistemi (V/I)

7.1. Splošno

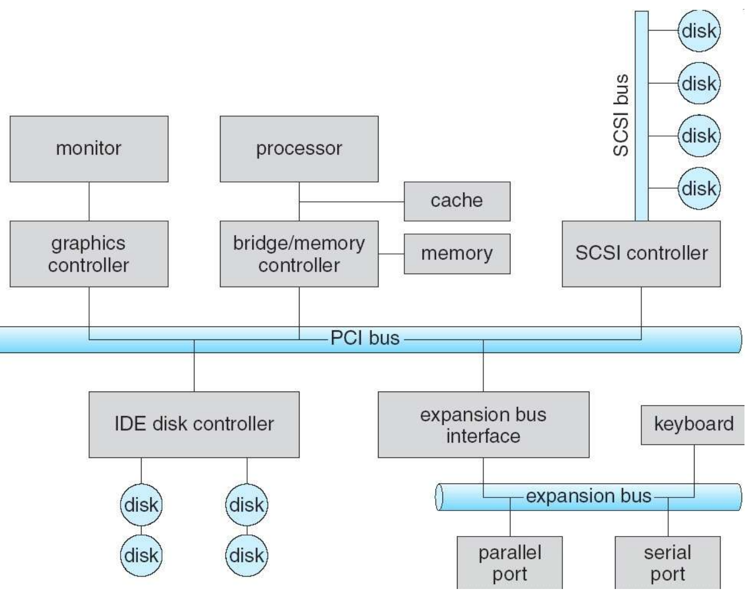
Dve glavni nalogi računalnika sta V/I operiranje in procesiranje. Ko na primer brkljamo po internetu ali urejamo datoteko naša glavna skrb je branje in vnašanje informacij. Vloga operacijskega sistema v kontekstu V/I operiranja je upravljanje in kontrola V/I operacij in V/I naprav.
Osnovni elementi strojne opreme so vrata, vodila in kontrolerji naprav gostijo velik spekter V/I naprav. Jedro operacijskega sistema je sestavljeno tako, da uporablja **gonilnike naprav**. Gonilniki naprav omogočajo unikatno komunikacijo z V/I podsistemom, tako kot sistemski klici omogočajo komunikacijo med uporabniškimi programi in operacijskim sistemom.

7.1.1. Vhodno / Izhodna strojna oprema

Delimo jih lahko na:

- naprave za shranjevanje (magnetni diski, magnetni trakovi, ...)
- prenosne naprave (mrežne kartice, modemi, ...)
- človeški vmesniki z računalnikom (ekran, tipkovnica, miška, ...)
- druge specializirane naprave (komandna ročica v letalih, ...)

Naprava komunicira z računalniškim sistemom tako, da pošlje signal po prevodniku ali celo zraku. Naprava komunicira z sistemom preko komunikacijske točke ali **vrat** (port) – na primer serijska vrata (serial port - COM1). Če ena ali več naprav uporablja skupen prevodnik za komunikacijo pravimo prevodniku **vodilo**(bus). Vodilo sestavlja set žic in strogo definiran protokol, ki definira nabor sporočil, ki se prenašajo po vodilu.
Vodila so pogosto uporabljena v računalniških arhitekturah.
Na sliki vidimo tipično PCI vodilo (common PC system bus), ki ga uporabljajo osebni računalniki. Vodilo povezuje procesorsko-pomnilniški podsistem s hitrimi napravami (grafična kartica, IDE in SCSI magnetni disk, ...) in **razširitvenim vodilom**, ki povezuje relativno počasne naprave (zaporedna in vzporedna vrata, ...) Vsaka naprava ima **kontroler**. Kontroler je skupek elektronike, ki operira z vrati, vodili in z napravo, ki jo kontrolira. Kontroler zaporednih vrat je enostaven kontroler, ki je na enem integriranem vezju in kontrolira signale na žicah, ki so povezane na zaporedna vrata. Na drugi strani imamo SCSI kontroler, ki ni enostaven. SCSI protokol je kompleksen in je zato kontroler izveden kot razširitvena kartica. Kartica vsebuje procesor, mikrokodo,



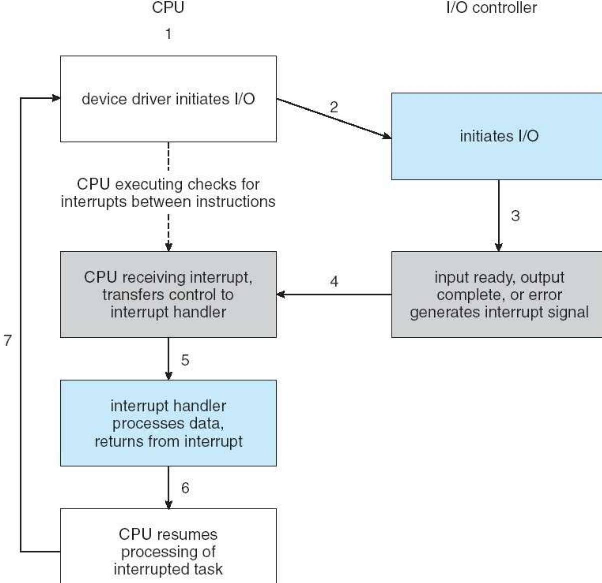
pomnilnik, ... Obstajajo tudi naprave, ki imajo kontroler vgrajen v napravo. ČE pogledamo v magnetni disk, vidimo, tiskano ploščico na eni strani. Ta ploščica je kontroler magnetnega diska. Kontroler diska, ki je v napravi vsebuje procesor, pomnilnik, mikrokodo, ki lokalno rešuje diagnosticiranje slabih blokov, predpomnjenje, medpomnjenje, ...
Kako procesor poda ukaz in podatke kontrolerju, da se izvede V/I prenos? Kontroler ima enega ali več registrov za kontrolne signale in podatke. Procesor komunicira z kontorlerjem tako, da bere in piše bite v te registre.
Po eni poti lahko pride do komunikacije preko posebne V/I instrukcije, ki določi prenos Byta ali besede na V/I **naslovu vrat**. V/I instrukcija sproži linije na vodilu, da izberejo pravo napravo in nato preberejo iz ali vpišejo v registre kontrolerja naprave.
Po drugi poti lahko V/I kontroler podpira **dodeljevanje v pomnilniku**. V tem primeru so kontrolni registri naprave povezani s pomnilniškim prostorom procesorja. Procesor izvede V/I zahtevo tako, da uporabi standardne instrukcije za branje in pisanje v kontrolne registre naprave.
Slika prikazuje običajen **naslovni prostor vrat** V/I naprave na osebnem računalniku. Kontroler grafične kartice ima V/I vrata za osnovne kontrolne operacije, kontroler sam, pa ima široko pomnilniško-mapirano območje za shrambo slike na ekranu. Proces pošlje informacijo v grafični kontroler tako, da zapiše podatek v pomnilniško-mapirano lokacijo. Kontroler generira sliko na podlagi podatkov v tem pomnilniškem prostoru. Seveda je pisanje milijonov Bytov v pomnilnik grafične kartice hitrejše kot izvajane milijon V/I instrukcij.

I/O address range (hexadecimal)	device
000–00F	DMA controller
020–021	interrupt controller
040–043	timer
200–20F	game controller
2F8–2FF	serial port (secondary)
320–32F	hard-disk controller
378–37F	parallel port
3D0–3DF	graphics controller
3F0–3F7	diskette-drive controller
3F8–3FF	serial port (primary)

V/I vrata so običajno sestavljena iz 4 registrov (statusni, kontrolni, data-in, data-out register).

7.1.2. Prekinitve

Osnovni prekinitveni mehanizem deluje takole:
Procesor ima linijo (fizično nožico), ki se imenuje **zahteva po prekinitvi** (interrupt request line). Procesor je na stanje te linije pozoren ob vsaki izvršeni instrukciji. Ko procesor zazna, da je kontroler povzročil signal na tej liniji, procesor shrani trenutna stanja registrov (stanje instrukcijskega kazalca, ...) in skoči na **prekinitveno rutino** (interrupt-handler routine) na točno določeno lokacijo v pomnilniku. Prekinitvena rutina ugotovi vzrok prekinitve, izvede potrebno procesiranje in vrne vrednost instrukcijskega kazalca, da procesor nadaljuje delo, ki ga je pred pred prekinitvijo izvajal. Prekinitvena rutina tudi pobriše signal v kontrolerju na prekinitveni liniji.



Večina novejših procesorjev ima dve prekinitveni liniji. **Maskirno** in **nemaskirno** linijo. S tem izbere kritične in nekritične prekinitve.
Nemaskirna prekinitvena linija prekinja procesor ob nepopravljivih napakah v pomnilniku (nujno je prekiniti).
Maskirno prekinitveno linijo uporabljajo kontrolerji naprav za normalno izvajanje V/I operacij. Maskirno prekinitev lahko procesor izključi, ko obdeluje prednostno nemaskirano prekinitev.

Slika prikazuje zgradbo **prekinitvenih vektorjev** Intel Pentium procesorja. Prekinitveni vektor vsebuje naslove v pomnilniku, kjer se prekinitvena rutina za določeno prekinitev začne.

vector number	description
0	divide error
1	debug exception
2	null interrupt
3	breakpoint
4	INTO-detected overflow
5	bound range exception
6	invalid opcode
7	device not available
8	double fault
9	coprocessor segment overrun (reserved)
10	invalid task state segment
11	segment not present
12	stack fault
13	general protection
14	page fault
15	(Intel reserved, do not use)
16	floating-point error
17	alignment check
18	machine check
19–31	(Intel reserved, do not use)
32–255	maskable interrupts

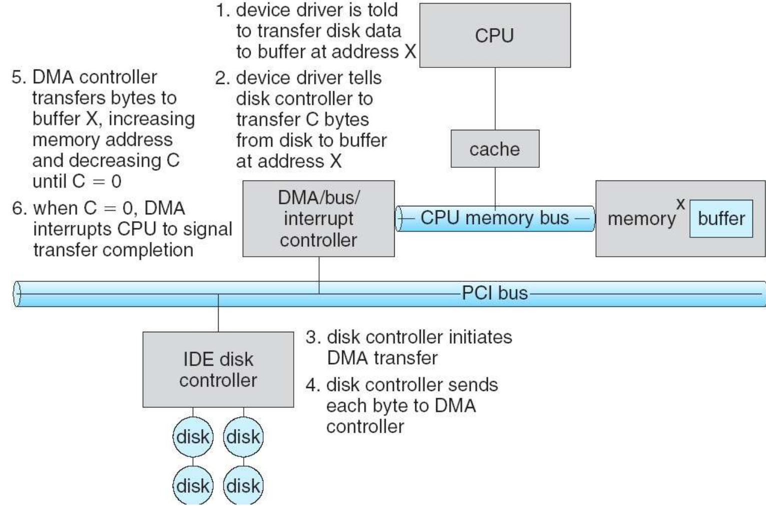
Dogodki od 0 do 31, ki so nemaskirni signalizirajo stanja nekaterih napak. Dogodki od 32 do 255, ki so maskirni so namenjeni streženju prekinitvam, ki jih povzročijo kontrolerji naprav.
Moderen operacijski sistem sodeluje z **prekinitvenim mehanizmom** na več načinov. Ob zagonu, operacijski sistem testira strojna vodila, da ugotovi katere naprave so prisotne in glede na prisotne naprave namesti pripadajočo prekinitveno rutino v tabelo prekinitvenih vektorjev.
Prekinitveni mehanizem obravnava tudi **izjeme** (exception), kot so deljenje z 0, dostop do zaščitene ali neobstoječe lokacije v pomnilniku ali poizkus izvedbe privilegirane instrukcije v uporabniškem načinu.
Prekinitveni mehanizem obravnava tudi **sistemske klice**, ki jih pošlje aplikacija, ko hoče posredovanje jedra. Sistemski klic preveri argumente, ki jih je aplikacija podala, zgradi

podatkovno strukturo za posredovanje argumentov jedru in nato izvede posebno instrukcijo – **programska prekinitev** ali **past** (trap). Ko sistemski klic izvede programsko prekinitev, prekinitvena strojna oprema shrani vrednosti uporabniške kode, preklopi na administrativen način in preda jedru rutino, ki implementira zahtevan servis. Programska prekinitev ima nižjo prioriteto kot strojna prekinitev, to pomeni, da izvajanje branja podatkov iz diska (strojna prekinitev) ne prekine programska prekinitev.

7.1.2.1. Direkten dostop do pomnilnika (DMA)

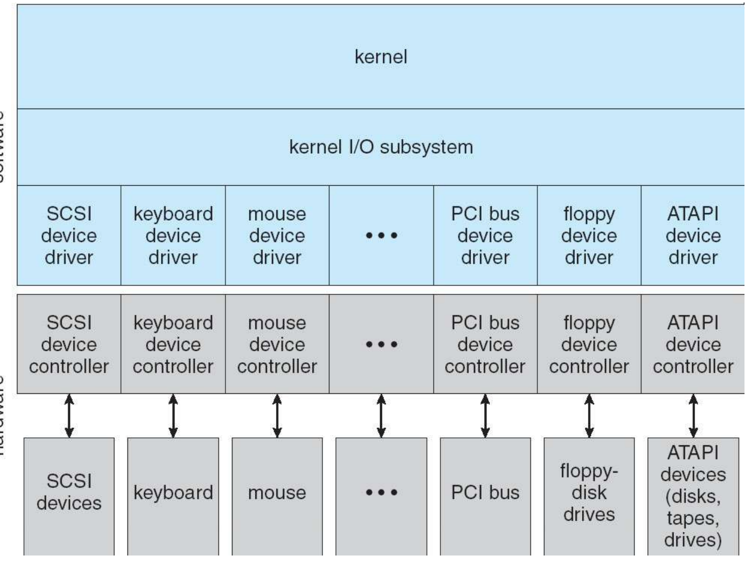
Za naprave, ki prenašajo velike količine Bytov kot je magnetni disk je zapravljanje procesorskega časa, če se konstantno kontrolira statusni bit in piše v podatkovni register Byte za Byte. Takemu načinu prenosa se reče **programiran V/I** (programed I/O – PIO način).
Veliko sistemov ne moti procesorja na PIO način prenosa podatkov, ampak preda to delo posebnemu kontrolerju, ki skrbi za direkten dostop do pomnilnika in se imenuje direct memory access – **DMA kontroler**.

Slika prikazuje korake pri DMA prenosu.



7.1.2.2. *Aplikacijski V/I vmesnik*

Poglejmo kako aplikacija lahko odpre datoteko na disku brez da bi vedela kakšne vrste je disk in kako lahko dodamo disk v sistem brez da bi to vplivalo na delovanje operacijskega sistema. To lahko dosežemo s programskimi nivoji.



Ločimo razlike V/I naprav tako, da definiramo posebnosti vsake V/I naprave posebej. Vsaka posebnost je dostopna preko standardnega niza funkcij – **vmesnika**. Posebnosti in razlike V/I naprav so vgrajene v module jedra imenovane **gonilniki naprav**. Cilj nivoja gonilnikov naprav je, da skrije razlike med kontrolerji naprav pred V/I podsistemom jedra.
Če torej naredimo V/I podsistem neodvisen od strojne opreme olajšamo delo programerjem in sistemskim razvijalcem!
Na nesrečo proizvajalcev strojne opreme, vsak tip operacijskega sistema vsebuje svoje standarde za gonilnike naprav. Vsaki strojni opremi morajo tako biti priloženi gonilniki za več operacijskih sistemov (Windows 95, 98, Me, NT, 2000, XP in Solaris).

Kot vidimo na sliki se naprave razlikujejo v več dimenzijah:

aspect	variation	example
data-transfer mode	character block	terminal disk
access method	sequential random	modem CD-ROM
transfer schedule	synchronous asynchronous	tape keyboard
sharing	dedicated sharable	tape keyboard
device speed	latency seek time transfer rate delay between operations	
I/O direction	read only write only read–write	CD-ROM graphics controller disk

- **znakovni** ali **blokovni** prenos podatkov – znakovni prenos podatkov prenaša byte za byte, medtem ko blokovni prenos prenaša bloke - niz bytov na enkrat.
- **zaporedni** ali **naključni** dostop do podatkov – zaporedni prenos podatkov v fiksnem zaporedju, ki ga določi naprava, medtem ko uporabnik naključnega dostopa lahko določi, da naprava išče na vsaki razpoložljivi lokaciji.
- **sinhrona** ali **asinhrona** naprava - sinhrona naprava je tista naprava, ki izvede prenos podatkov v naprej določenem odzivnem času, medtem ko je asinhrona naprava tista naprava, kjer ne vemo koliko bo odzivni čas naprave pri prenosu podatkov.
- **deljiva** ali **nedeljiva** naprava – deljiva naprava je tista, ki dopusti, da jo uporablja več procesov
- **hitrost operacij** – hitrost operacij V/I naprav varirajo od nekaj B/s do nekaj GB/s.
- **vrsta dostopa** – branje iz in pisanje v napravo (dvosmerne naprave), samo za branje

ter samo za pisanje (enosmerne naprave)

7.1.2.3. V/I podsistem jedra

Jedro omogoča veliko servisov, ki so povezani z V/I operacijami:

- **V/I razvrščanje** – več zahtev po V/I operacijah je treba pravilno in pravično postaviti v vrsto (vzdržuje se čakalna vrsta V/I naprav)
- **buffering** – buffer je del pomnilnika namenjen izmenjavanju informacij med dvema napravama ali napravo in aplikacijo. Kreiranje bufferja se vidi v:
  - neenakomerna hitrost dveh naprav (shranjevanje datoteke preko modema na disk, ko je buffer poln, se celotna vsebina bufferja shrani na disk v eni operaciji)
  - prilagajanje dveh naprav, ki imata različne velikosti prenosa podatkov (delov omrežju, kjer so bufferji uporabijo za drobljenje in ponovno sestavljanje sporočil – na strani oddajnika se sporočilo razdrobi v majhne pakete – mrežne pakete, na strani sprejemnika se paketi sestavijo v sestavljalnem bufferju in tvorijo sliko originalnega sporočila).
- **medpomnjenje** – medpomnilnik je področje hitrega spomina, ki drži kopije podatkov. Dostop do podatkov v medpomnilniku je bolj efektivno kot do originalnih podatkov. Instrukcija trenutno tekočega procesa je shranjene na disku, kopirane v fizičnem pomnilniku in v hitrem primarnem in sekundarnem pomnilniku procesorja.
- **spooling** – spool je buffer, ki vsebuje izhod za napravo (tiskalnik in fax ne more istočasno tiskati več izpisov). Zato operacijski sistem prestreže izhode procesov, ki pošiljajo podatke tiskalniku in jih shrani v ločeno datoteko na disku. Ko ena aplikacija konča tiskanje "spooling" sistem procesu, ki je na vrsti dodeli tiskalnik.

7.2. Vprašanja za ponavljanje

1. Opiši kaj se dogaja, ko kontroler pošlje signal po prekinitveni liniji procesorja.

2. Naštej v katerih dimenzijah se lahko razlikuje strojna oprema in naštej vsaj en primer pri vsaki lastnosti.

3. Kakšna je vloga maskirne in nemaskirne prekinitvene linije?

4. Zakaj služi in kaj počne DMA kontroler?

8. Viri in literatura

Literatura:
Saša Divjak: Operacijski sistemi, FRI Maj 2001
Ida M. Flinn, Ann McIver McHoesoesUnderstanding operating system
Silbertshatz, Galvin, Gagne: Operating system Concepts 7th edition

Internet:
http://colos1.fri.uni-lj.si/~sis/computing/OPERACIJSKI_SISTEMI/index.html
http://www.netnam.vn/unescocourse/os/os_frm.htm
http://ai.ijs.si/MarkoBohanec/Informatika/InfTeh.htm
http://www.vus.uni-lj.si/kat_org-inf/inf/OIK_P_P_infpedpg.htm

http://storm.uni-mb.si/vaje/os_vss/

http://www.osdata.com/

http://williamstallings.com/OS4e.html

Operating systems concepts:
http://cs-www.cs.yale.edu/homes/avi/os-book/os7/slide-dir/index.html
http://he-cda.wiley.com/WileyCDA/HigherEdMultiTitle.rdr?name=silberschatz

Wikipedia:
http://en.wikipedia.org/wiki/List_of_operating_systems
http://en.wikipedia.org/wiki/Comparison_of_operating_systems
http://en.wikipedia.org/wiki/Comparison_of_file_systems

http://www.infocom.cqu.edu.au/Courses/aut2001/85349/Resources/Lectures/

Malo nižji nivo razlage OS:
http://physinfo.ulb.ac.be/cit_courseware/opsys/ostart.htm