Mäluhaldus

Ülevaade

- Põhimõisted
- Pidevate mälualade hõivamine
- Lehekülgede saalimine
- Segmenteerimine
- Segmenteerimine koos lehekülgede saalimisega

Aadresside sidumine

Keelekonstruktsioonide sidumine mäluaadressidega võib toimuda erinevatel etappidel:

- Kompileerimise ajal — mälupaigutus on ette teada, genereeritakse kood absoluutaadressidega, paigutuse muutumisel vaja ringi kompileerida
- Laadimise ajal — genereeritakse mälus ümberpaigutatav kood, reaalsed mäluaadressid asendatakse programmi laadimisel
- Täitmise ajal — aadresside sidumine lükatakse edasi konkreetse aadressi poole pöördumiseni. Programmi saab täitmise ajal mälus ringi liigutada. Vajalik riistvaraline tugi.

Loogilised ja füüsilised aadressid

- Loogilise ja füüsilise aadressiruumi eraldamine on mäluhalduses üks olulisemaid kontseptsioone
  - Loogiline aadress — programmide poolt kasutatatav aadressiruum, nimetatakse ka virtuaalseks aadressiruumiks
  - Füüsiline aadress — aadress, mida mäluseade tegelikult näeb ja kasutab
- Kompileerimise ja laadimise ajal seotavate aadresside puhul on loogiline ja füüsiline aadress samad, täitmise ajal seotavatel erinevad
- MMU (*Memory Management Unit*) — riistavaraline seade loogiliste aadresside füüsiliseks teisendamiseks
- Kasutajaprogramm tegeleb oma loogiliste aadressidega (0..max) ega näe otseselt füüsilisi aadresse

Dünaamiline laadimine

- Alamprogramme ei laadita enne, kui neid kord vaja läheb
- Efektiivsem mälukasutus
- Kasulik näiteks siis, kui harva esinevate erijuhtude töötlemiseks on kokku palju koodi
- Ei vaja operatsioonisüsteemi tuge, realiseeritav täiesti programmi enda tasemel

Dünaamiline linkimine

- Linkimine lükatakse edasi täitmisajaks
- Tegeliku alamprogrammi asemel on proksi (*stub*), mis esimesel tema poole pöördumisel laadib päris alamprogrammi ja asendab ennast selle alamprogrammiga (enamasti viida vahetusega)
- Operatsioonisüsteemi tuge on vaja juhul, kui tahame neid alamprogramme jagada mitme protsessi vahel ja kasutame seejuures mälukaitset
- Eriti kasulik (süsteemsete) teekide kasutamisel — kettaruumi ja mälu kokkuhoid, mugav teekide uuendamine

Ülekatmine (*overlay*)

- Rakendusprogramm hoiab mälus ainult osa programmist ja andmetest, mida hetkel vaja on
- Kasutatakse, kui kogu programm kokku on suurem kui talle antud mälu
- Opsüsteemilt tuge pole vaja, realiseeritav kasutaja tasemel
- Näide: DOS ja suured programmid

Saalimine (*swapping*)

- Protsessi võib ajutiselt mälust välja salvestusruumi kirjutada ning hiljem tagasi mällu laadida
- Salvestusruum — kiire ketas, mis mahutab kõigi kasutajaprogrammide mälukujutised, peab olema otsejuurdepääsuga suvalise mälukujutise juurde
- *Roll out, roll in* — saalimise variant prioriteedil põhinevate planeerimisalgoritmide juurde: madalama prioriteediga protsess saalitakse välja kõrgema prioriteediga protsessi töö ajaks
- Enamuse saalimise ajast võtab andmete kopeerimine välisseademele, aeg on lineaarselt sõltuv mäluhulgast.
- Erineval kujul on saalimine kasutusel paljudes süsteemides, näiteks UNIX, Linux, Windows.

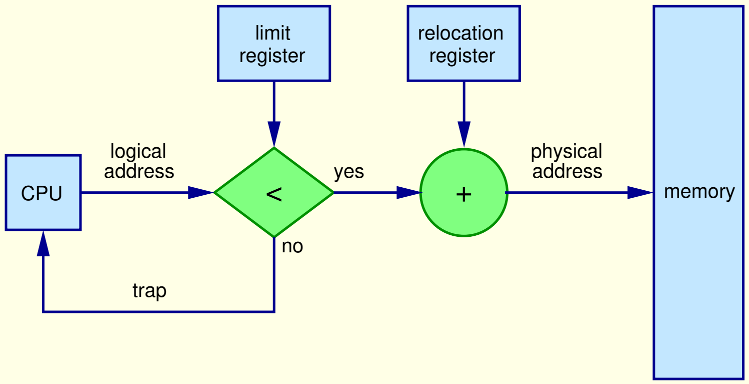
Pidevate mälualade hõivamine (1)

- Põhimälu on enamasti jagatud kahte ossa:
  - Residentne opsüsteem (tihti madalamatel mäluaadressidel katkestusvektori asukoha tõttu)
  - Kasutajaprotsessid
- Kasutajaprotsessidele mõeldud mälu on omakorda jaotatud väiksemateks tükkideks
- Igale täidetavale protsessile eraldatakse üks sobiva suurusega terviklik mälutükk
- Auk — vaba mälutükk (uued protsessid saavad mälu nendest)

Pidevate mälualade hõivamine (2)

Iga protsessiga on seotud teisaldus- ja piirregister

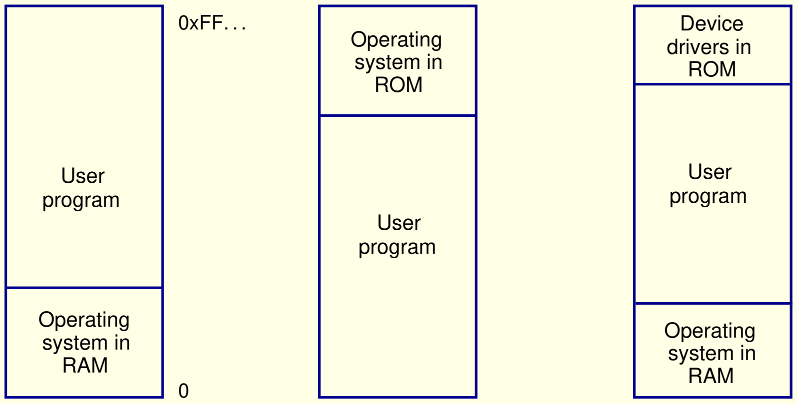
- Kasutatakse loogiliste aadressite teisendamisel
- Kaitseb teisi protsesse ja opsüsteemi vigaste mälupöörduste eest



Pidevate mälualade hõivamine (3)

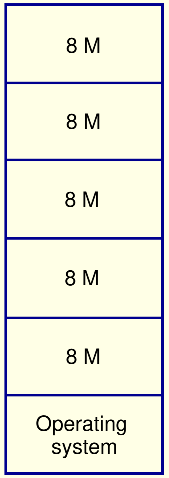
- Kasutajaprotsessidele mõeldud mälu tükkideks jagamine võib toimuda staatiliselt või dünaamiliselt
- Staatilise tükelduse korral on tükkide arv ja suurus eelnevalt fikseeritud
  - Kõik tükid võrdse suurusega
  - Erineva suurusega tükid
- Erijuht — monoprogrammeerimine, kus korraga täidetav ainult üks protsess, millele antakse kogu opsüsteemist ülejääv mälu

Mälujaotus monoprogrammeerimise süsteemides



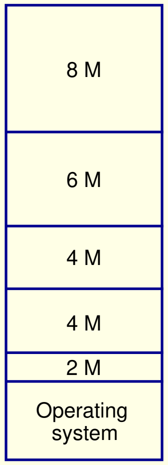
Fikseeritud tükeldusega mälujaotus (1)

- Võrdse suurusega tükid
  - Protsess, mille suurus ei ületa tüki suurust, laetakse suvalisse vabasse tükki
  - Kui kõik tükid on hõivatud, võib opsüsteem mõne protsessi välja saalida
  - Kui programm ei mahu tükki ära, peab programmeerija kasutama ülekatmist
- Lihtne realiseerida, kuid mälukasutus on ebaefektiivne
  - Sisemine fragmenteerumine — ükskõik kui väike ka protsess ei oleks, võtab ta enda alla terve tüki



Fikseeritud tükeldusega mälujaotus (2)

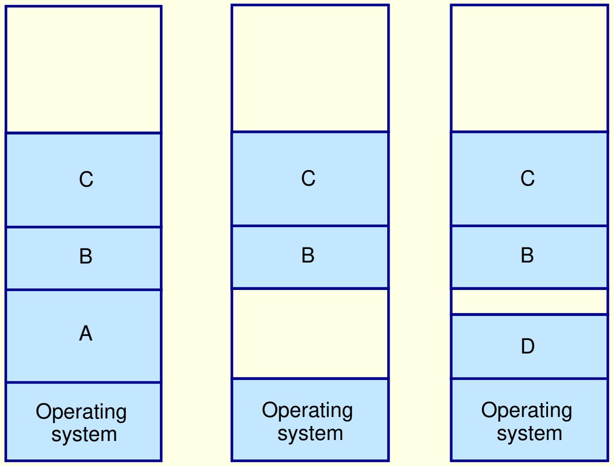
- Erineva suurusega tükid
  - Protsessile antakse vähima suurusega tükk, millesse ta ära mahub
- Igal tükil (või sama suurusega tükkidel) üks eraldi sisendjärjekord
  - Vähendab sisemist fragmenteerimist
  - Ebaefektiivne mitmeteguselisus
- Üks globaalne sisendjärjekord
  - Valida esimene protsess, mis auku ära mahub
  - Valida suurim protsess, mis auku ära mahub
  - Teisel juhul näljutamisoht



Dünaamilise tükeldusega mälujaotus (1)

- Tükkide arv ja suurus ei ole eelnevalt fikseeritud, vaid muutub vastavalt täidetavate protsesside arvule ja suurusele
- Protsessile antakse täpselt nii suur tükk, kui ta vajab
- Protsessi lõpetamisel või välja saalimisel vabanev tükk ühendatakse võimalusel ümbritsevate aukudega suuremaks auguks
- Kalduvus tekitada kasutuses olevate tükkide vahele väikeseid auke
- Väline fragmenteerumine — vajalik vaba mälu leidub, aga ta ei ole sidus

Dünaamilise tükeldusega mälujaotus (2)



Dünaamiline tükeldusega mälujaotus (3)

- Probleem: kuidas rahuldada päringut mäluplokile pikkusega *n* ettenatud aukude nimekirjast?
  - Esimene sobiv (*first-fit*) — kasutatakse esimest piisava suurusega auku. Esimest võib lugeda algusest või eelmisest leitud august alates
  - Parim sobiv (*best-fit*) — kasutatakse vähimat piisava suurusega auku. Tuleb läbi otsida kogu nimekiri, enamasti on see suuruse järgi järjestatud. Vähimad ülejäävad tükid
  - Halvim sobiv (*worst-fit*) — kasutatakse suurimat vaba auku, samuti vaja kogu nimekiri läbi vaadata. Suurimad ülejäävad tükid.
- Esimene sobiv ja parim sobiv annavad parema kiiruse ja mälukasutuse kui halvim sobiv

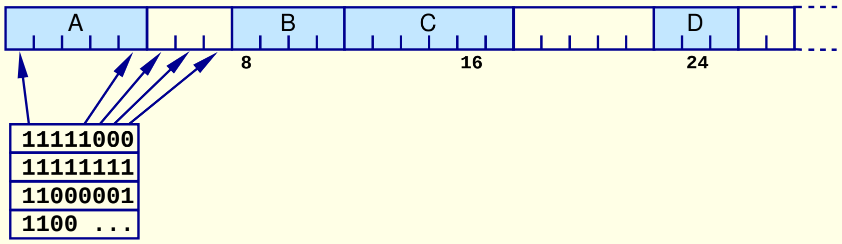
Fragmenteerumine ja tihendamine

Välist fragmenteerumist saab vähendada mälu tihendamise (*compaction*) abil

- Tõstame hõivatud mälualasid ümber nii, et kogu vaba mälu oleks kompaktselt koos
  - Näiteks liigutame protsessid kõik mälu algusse üksteise järele
- Võimalik ainult dünaamilise teisaldamise korral
- Pooleli olev I/O segab (kopeerime OS puhvritesse või jätame protsessi paigale)
- Suhteliselt kulukas

Mäluhaldus bititabeliga

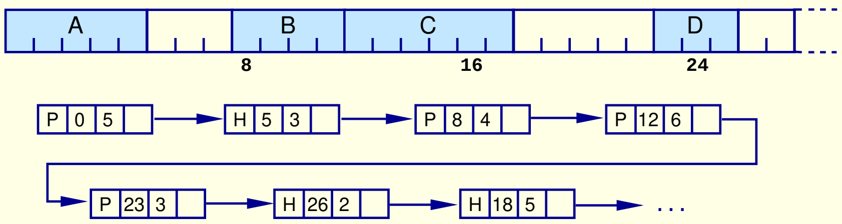
- Süsteem peab arvet pidama hõivatud ja vabade mälupiirkondade vahel
- Lihtsaim meetod on kasutada bititabelit



- Kõrvuti olevate aukude ühendamine automaatne
- Piisava suurusega vaba piirkonna leidmine aeglane

Mäluhaldus listidega (1)

- Iga tüki jaoks peame meeles tema staatuse, alguse ja pikkuse
- Vastava info hoiame dünaamilises listis



- Kõrvuti olevate aukude ühendamine lihtne
- Piisava suurusega vaba piirkonna leidmine kiirem kui bititabeli korral, kuid siiski linaarse keerukusega

Mäluhaldus listidega (2)

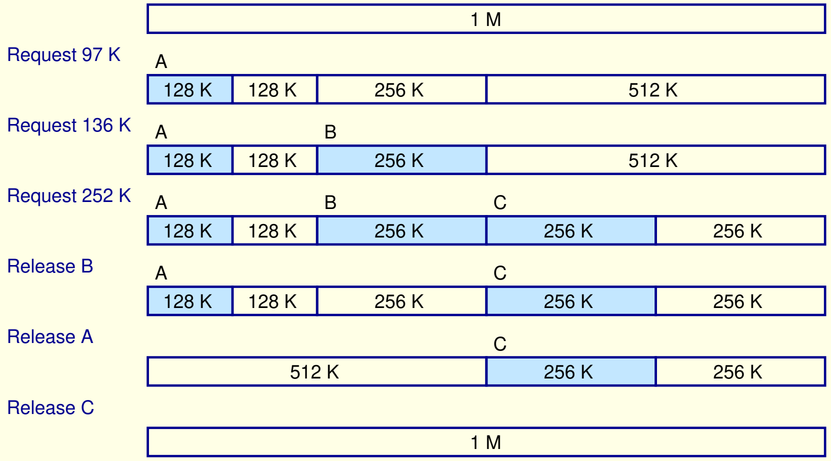
Variatsioonid:

- Haldame eraldi listi vabade ja hõivatud tükkide kohta
  - Piisava suurusega vaba tüki leidmiseks pole vaja enam hõivatud tükke inspekteerida
  - Pole vajadust enam staatusbiti (P/H) järele
  - Vabade tükkide listi jaoks pole eraldi mälu vaja, kuna neid tükke saab ise selleks ära kasutada!
  - Kõrvuti olevate aukude ühendamine on suhteliselt kulukas
- Haldame sagedasti vajaminevate tükisuuruste jaoks eraldi vabade tükkide listi
  - Piisava suurusega vaba tüki leidmine kiire

”Buddy” süsteem

- Kogu olemasolevat mälu käsitletakse kui üht vaba mälutükki suurusega 2^U
- Kui küsitava mälu suurus on $2^{U-1} \leq n \leq 2^U$, siis hõivatakse kogu tükk
- Vastasel korral jaotatakse tükk kaheks võrdseks osaks ja kogu protsessi korratakse kuni sobiva suurusega tüki tekkimiseni
- Tüki vabastamisel kontrollitakse, kas ”kaastükk” on vaba
  - Kui ei ole, siis lisatakse tükk antud suurusega vabade tükkide listi
  - Vastasel korral tükid ühendatakse ning tekkinud tükki püütakse omakorda ühendada oma ”kaastükiga”

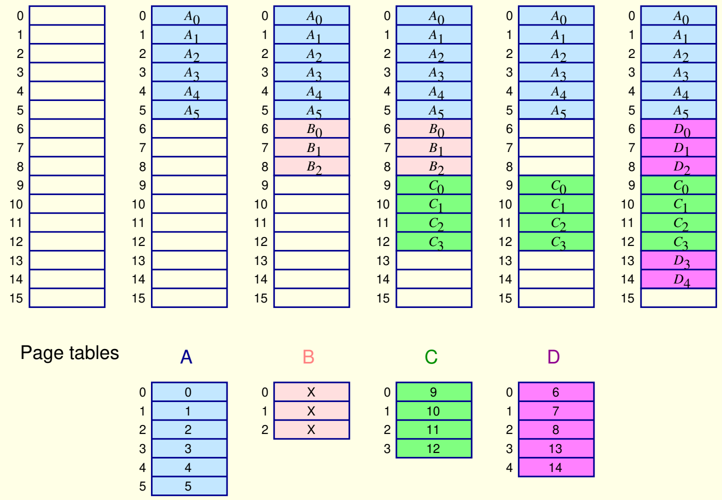
”Buddy” süsteemi näide



Lehekülgede saalimine (*paging*)

- Protsessi füüsiline aadressiruum ei pea tingimata olema pidev — protsessi eri osad võivad asuda füüsilises mälus suvaliste kohtade peal laiali
- Jagame füüsilise mälu fikseeritud suurusega lehekülgedeks (kaadriteks — *frame*), suurus kahe aste (näiteks 512, 4096, 8192)
- Jagame loogilise mälu samasugusteks lehekülgedeks (*page*)
- Peame arvet vabade kaadrite üle
- Suurusega n protsessi jaoks otsime lihtsalt n vaba kaadrit
- Iga protsessi kohta teeme lehekülgede tabeli loogiliste lehekülgede ja füüsiliste kaadrite vastavusse seadmiseks
- NB! Lehekülgede kaupa hõivamisel tekib paratamatult sisemine fragmenteerumine

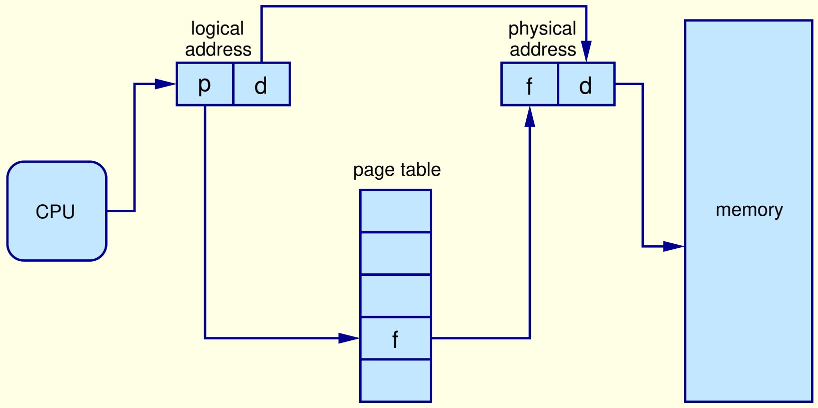
Mälu eraldamine lehekülgede kaupa



Aadresside tõlkimine (1)

- Mäluaadressi bitid jagatakse kahte gruppi:
  - Lehekülje number (p) — kasutatakse indeksina leheküljetablisse lehekülje füüsilise aadressi leidmiseks
  - Nihe lehekülje sees (d) — liidetakse lehekülje füüsilisele aadressile täieliku füüsilise aadressi saamiseks
- Kontekstivahetuse ajal programmeerimine MMU vastavalt protsessi leheküljetabelile

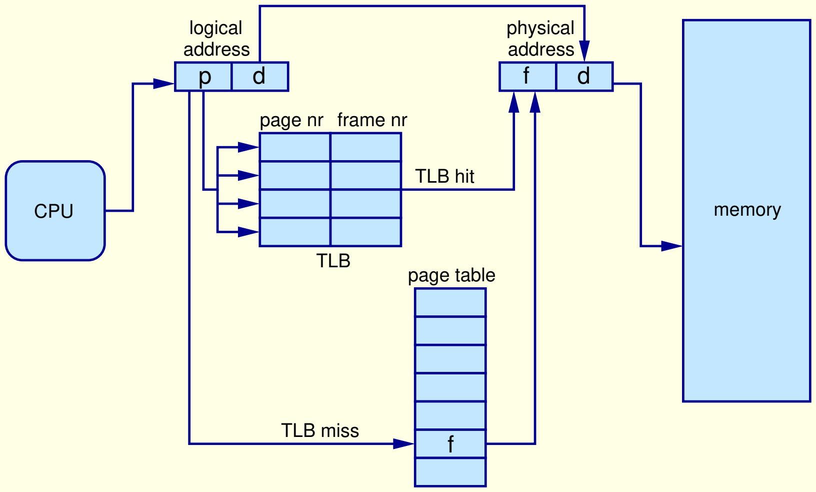
Aadresside tõlkimine (2)



Leheküljetabelite realiseerimine

- Leheküljetabeleid hoitakse mälus
- Leheküljetabeli algust näitab vastav register (PTBR)
- Leheküljetabeli pikkust näitab kah vahel register (PTLR)
- Iga soovitav mälupöördus nõuaks kahte tegelikku mälupöördust (leheküljetabeli kirje + tegelik pöördus tõlgitud aadressil)
- Seda probleem lahendab viimati- (sagedamini?) kasutatud kirjete puhverdamine protsessori registrites
- TLB (*Translation Lookaside Buffer*) — assotsiatiivmälu abil realiseeritud leheküljetabeli kirjete puhver
- TLB *miss* — alati ei leita vastavat kirjet TLB-st, sel juhul tuleb ta mälust TLB-sse lugeda
- ASID (*Address Space ID*), TLB *flush*

Aadresside tõlkimine TLB abil



Efektiivne mälupöördusaeg

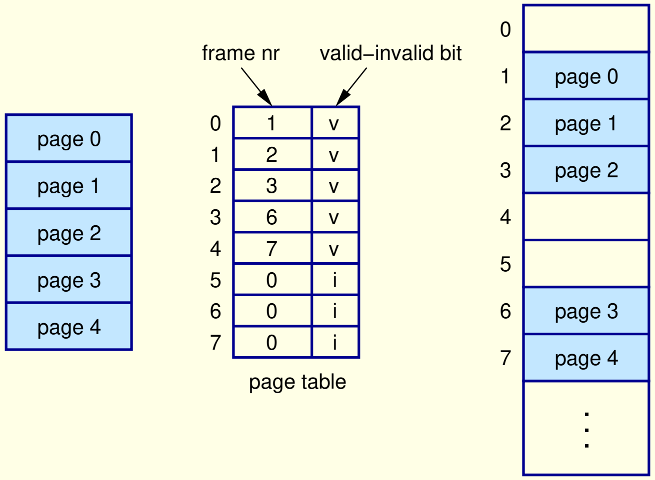
- Reaalne mälupöördus = 1 ajaühik
- Assotsiatiivmälust otsimine = ε ajaühikut
- Assotsiatiivmälu edukate otsimiste osakaal (*hit ratio*) — mitmel protsendil juhtudest leiab tulemuse assotsiatiivmälust. Tähistame α
- Efektiivne mälupöörduse aeg:

$$EAT = (1+\varepsilon)\alpha + (2+\varepsilon)(1-\alpha) = 2+\varepsilon-\alpha$$

Mälukaitse (1)

- Mälukaitset saab (ja on mõtet) siduda iga leheküljega
- Iga lehekülje juurde paneme lipu *valid*
  - *valid* — lehekülg on protsessi mälupiirkonnas
  - *invalid* — lehekülge ei ole protsessi mälupiirkonnas
- Näiteks mälupiirkonna lõpu tähistamiseks

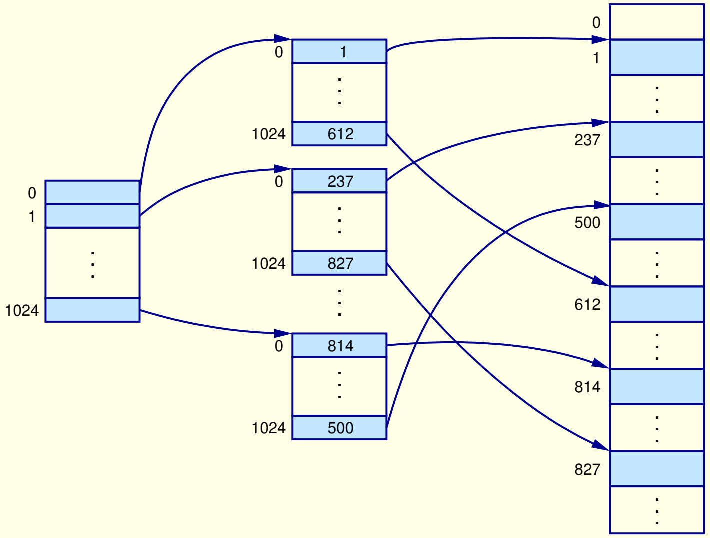
Mälukaitse (2)



Hierarhilised leheküljetabelid

- Kuna üks leheküljetabel läheks liiga suureks, teeme leheküljetabeli mitmetasemelise (hierarhilise)
- Kahetasemeline leheküljetabel — aadress jagatakse leheküljesiseseks aadressiks ning lehekülje number omakorda kaheks bitigrupiks (kummagi taseme jaoks üks grupp, määrab indeksi vastavas tabelis)
- Näiteks 32-bitine aadress, 4K lehekülg (12 bitti), ülejäänud 20 bitti jagatakse kaheks 10-bitiseks leheküljenumbriks (välimise leheküljetabeli indeks ja sisemise leheküljetabeli indeks)
- Füüsilised leheküljed võivad endiselt olla läbisegi (sõltumata kummagi tabeli järjekorrast)
- Kolmetasemelised, neljatasemelised, . . .

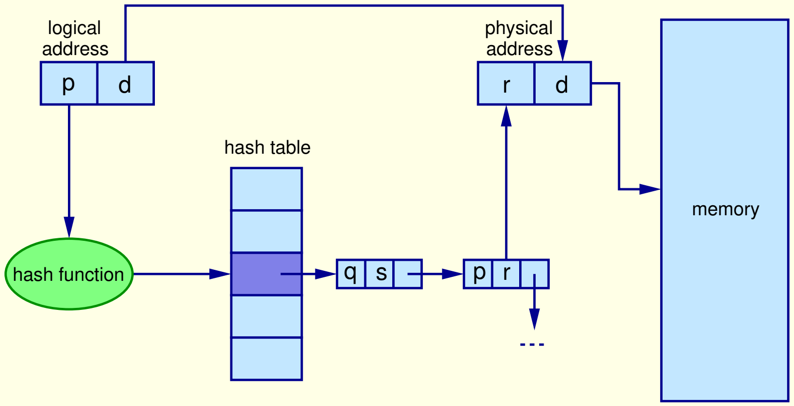
Kahetasemeline leheküljetabel



Paisksalvestusega leheküljetabelid (1)

- 64-bitisel süsteemil tuleks palju tasemeid leheküljetabeleid, pole efektiivne
- Virtuaalse lehekülje number salvestakse paiskfunktsiooni abil leheküljetabelisse
- Leheküljetabel on välisahelatega paisktabel (mitu erinevat lehekülge võivad samale positsioonile sattuda)
- Kasutamine kiire
- Kontekstivahetus aeglane

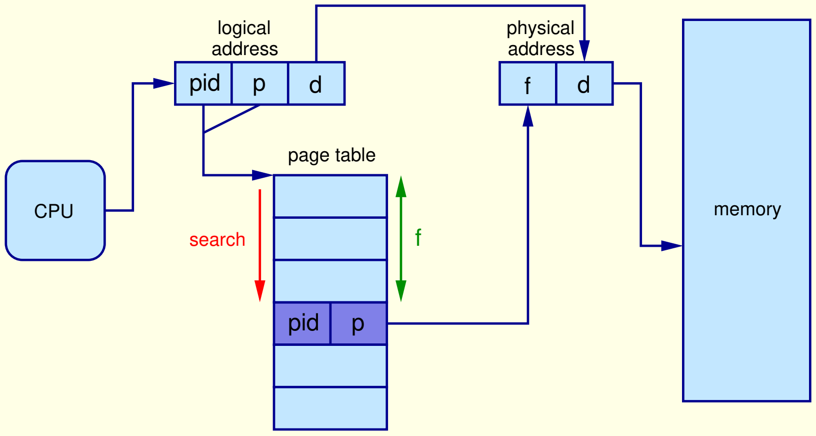
Paisksalvestusega leheküljetabelid (2)



Pööratud leheküljetabel (1)

- Idee: leheküljetabelis on kirje iga füüsilise mälu lehekülje kohta
- Kokku 1 tabel, mitte igal protsessil oma tabel, mida siis vahetada tuleks
- Füüsilise mälu leheküljele vastav tabelikirje koosneb protsessi identifikaatorist ja protsessisisesest loogilisest aadressist
- Mälutarve väiksem
- Otsimisaeg suurem
- Otsimise kiirendamiseks paisktabel leheküljetabeli ees (muidugi ka TLB)

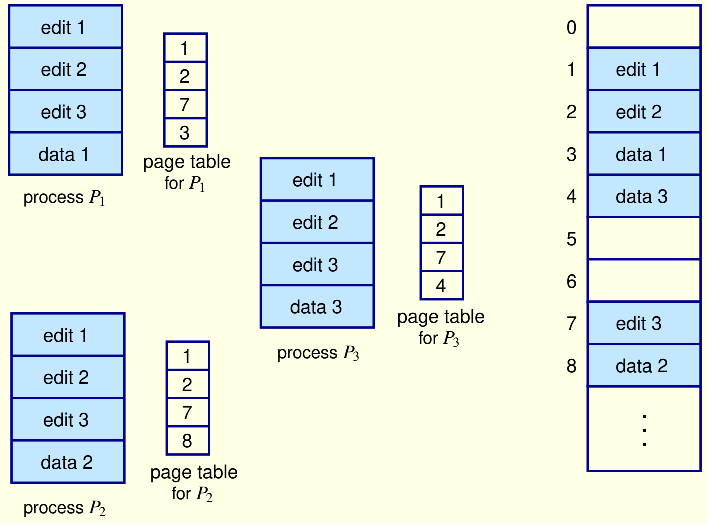
Pööratud leheküljetabel (2)



Jagatud mäluleheküljed (1)

- Koodi sisaldavaid lehekülgi on mõtet jagada
- Võimalik siis, kui kood on taassisenetav (*reentrant*) — ei modifitseeri iseennast
- Jagatud lehekülg on nähtav mitme protsessi aadressruumist
- Sissekirjutatud aadressidega kood peab igas protsessis samal virtuaalaadressil asuma
- Positsioonist sõltumatu kood (PIC — *Position Independent Code*) võib igas protsessis olla erineva virtuaalaadressi peal
- Uuemal ajal on jagatud teegid enamasti positsioonist sõltumatud
- Igal protsessil oma andmed ja enamasti ka mingi osa mittejagatud koodi
- Pööratud leheküljetabelite puhul on lehekülgede jagamine raskem

Jagatud mäluleheküljed (2)



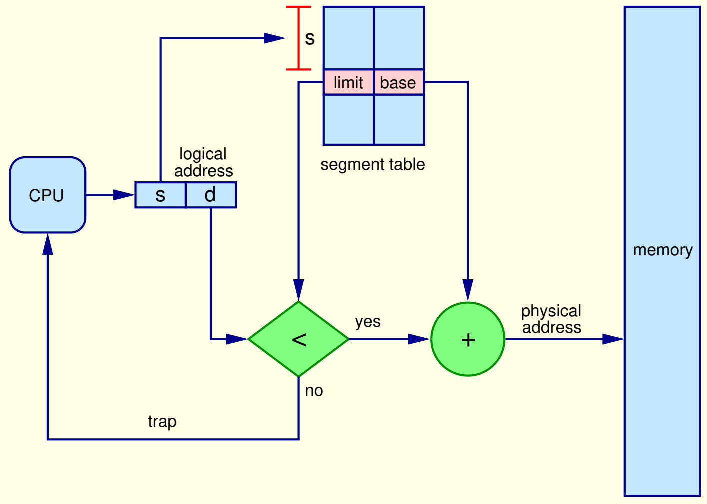
Segmenteerimine

- Segmenteerimine on mäluhalduse skeem, kus ei kasutata mitte ühte suurt pidevat loogilist aadressruumi, vaid paljusid segmente
- Läheb paremini kokku paljude kasutajate mõttemaailmaga
- Programm on komplekt segmente. Segment on loogiline ühik, näiteks
  - Põhiprogramm
  - Protseduur, funktsioon, meetod
  - Objekt
  - Lokaalsed muutujad, globaalsed muutujad
  - Magasin
  - ...

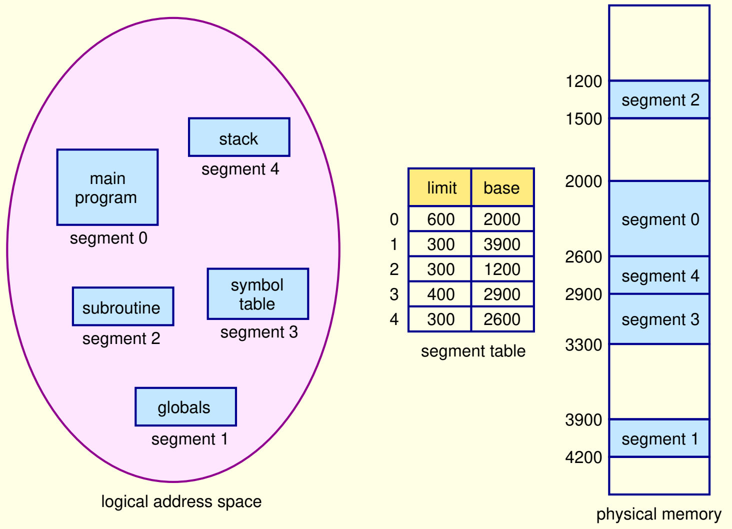
Segmenteerimisega arhitektuur

- Loogiline aadress on paar <segmendi number, nihe>
- Segmenditabel — seab igale segmendi numbrile vastavusse mälutüki:
  - Baas — segmendi alguse füüsiline aadress
  - Limiit — segmendi pikkus
- Segmenditabeli algus näidatakse vastava registriga (STBR)
- Segmenditabeli pikkus kah enamasti registris (STLR) — sisuliselt on tegemist maksimaalse kasutatava segmendi numbriga
- Mälu jagatakse tervete segmentidega kaupa (first-fit, best-fit näiteks), seega tekib väline fragmenteerumine
- Segmente saab protsesside vahel jagada

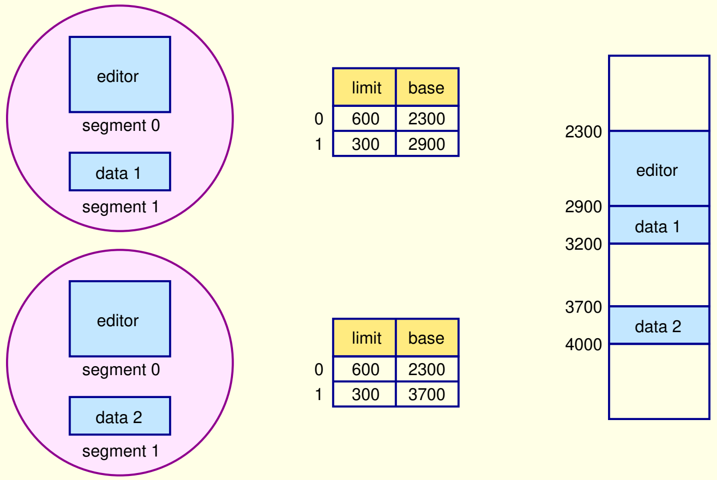
Aadrerssite teisendamine segmenteerimisel



Näide segmenteerimisest



Segmentide jagamine



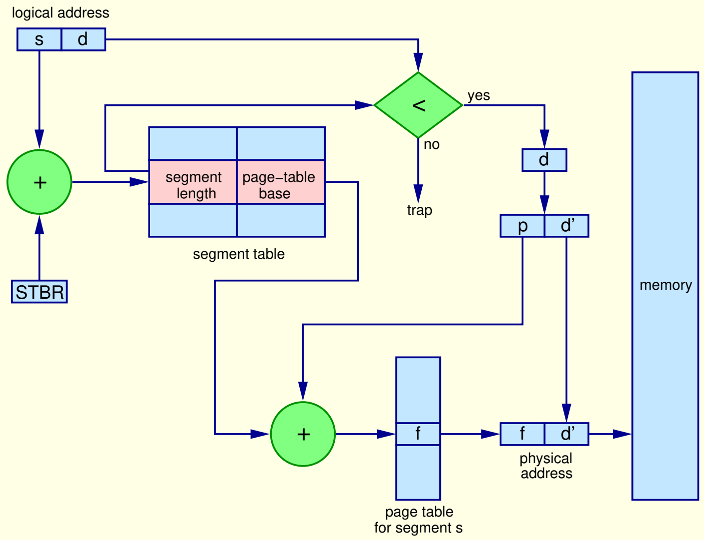
Segmenteerimine ja mälukaitse

- Mälukaitse läheb üsna loomulikku rada — enamasti on tervel segmendil sama kaitsetase, seega seome loabiti(d) segmenditabeli kirjega:
  - *valid* — kas segment on kasutusel
  - lugemise/kirjutamise/koodi täitmise lubamise bitid (suvalises kombinatsioonis)

Segmenteerimine koos lehekülgedega

- MULTICS lahendas välise fragmenteerumise probleemi segmenditabelite lehekülgedeks jagamisega — iga segmenditabeli kirje oli viide vastava segmendi leheküljetabelile
- Sarnane kombineeritud lähenemine on tänapäevalgi kasutusel
- Inteli 386 (tegelt kogu edasine x86 haru) kasutab kah segmenteerimist ja lehekülgi mõlemaid, kuid üksteise järel: segmeditabelist saadakse tulemusena mingi aadress ning sellele rakendatakse kahetasemelist leheküljetabelit
- Mõlema head küljed ära kasutatavad, natukese mälu ja keerukuse (OS ja protsessor) hinnaga

Segmenteerimine koos lehekülgedega MULTICS-is



Segmenteerimine koos lehekülgedega i386-s