

Virtuaalmälu

Ülevaade

- Taust
- Lehekülgede laadimine nõudmisel
- Lehekülgede asendamine
- Leheküljeasenduse algoritmid
- Füüsilise mälu haldus
- Pekslemine (*trashing*)

Taust

- Virtuaalmälu — kasutaja loogilise mälu füüsilisest mälust lahti sidumine
 - Ainult osa protsessi poolt vajatavast mälust on korraga füüsilises mälus
 - Loogiline aadressiruum saab seega olla oluliselt suurem kui füüsiline aadressiruum
 - Aadressirume (aadressiruumide osi) saavad protsessid omavahel jagada
 - Võimaldab efektiivsemalt protsesse luua
- Virtuaalmälu saab realiseerida kahel viisil
 - Lehekülgede laadimine nõudmisel (*demand paging*)
 - Segmentide laadimine nõudmisel (*demand segmentation*)

Lehekülgede laadimine nõudmisel

- Iga lehekülg laaditakse mällu alles siis, kui seda esimest korda vaja läheb
 - Vähem I/O pöördusi
 - Vähem mälu vaja
 - Kiirem reaktsiooniaeg
 - Rohkem protsesse
- Lehekülge vajatakse, kui keegi selle lehekülje poole pöördub
 - Vigane lehekülje aadress $\Rightarrow$ lõpetame protsessi
 - Lehekülg pole mälus $\Rightarrow$ toome ta mällu ja jätkame
- Sarnane lehekülgede saalimisele, aga „laisem“ mällu toomisel

Kehtivuse lipp

- Kuidagi on riistvaraliselt vaja vahet teha, kas lehekülg on mälus või mitte
- Seome iga leheküljega kehtivuse lipu (*valid-invalid flag*)
- 1 — mälus, 0 — pole mälus
- Algul kõigil lehekülgedel 0 (algul ei too üldse ühtegi lehekülge mällu)
- Aadresside tõlkimisel vaatab riistvara seda bitti
 - Kui kehtivuse lipp on 0, siis genereerib riistvara leheküljele pöördumise vea (*page fault*)

Leheküljele pöördumise viga

- Kui mingi lehekülje poole üldse kunagi pöördutakse, siis esimene pöördumine genereerib leheküljele pöördumise vea (püünise)
- OS vaatab oma leheküljetabelit ja otsustab, kas lehekülge polegi olemas või tuleb see lihtsalt mällu tuua
- Mällu toomine:
 - Otsida vaba füüsilise mälu kaader
 - Lugeda lehekülg kettalt sellesse kaadrisse
 - Panna leheküljetabelis vastava lehekülje kohta kehtivuse lipp püsti
 - Taaskäivitada viimane protsessorikäsk (see, mis püünise genereeris) — raskem kui pealtnäha tundub
 - * Mäluploki liigutamine
 - * Autoinkrement
 - * Üks käsk võib mitu püünist genereerida enne edukat täitmist

Lehekülgede asendamine

- Mis saab, kui pole ühtegi vaba füüsilise mälu kaadrit?
- Leida mälulehekülg, mis pole aktiivselt kasutusel
- Saalida see lehekülg välja
- Saalida selle asemele vajalik lehekülg
- $\Rightarrow$ lehekülgede asendamine
 - Mis algoritmi järgi valida asendatav lehekülg?
 - Kuidas on selle algoritmi jõudlusega?
- Sama lehekülge võidakse mällu tuua palju kordi

Nõudmisel laadimise jõudlus

- *Page Fault*'ide osakaal p ($0 \leq p \leq 1$)
 - $p = 0 \implies$ pole leheküljepöörduse vigu
 - $p = 1 \implies$ iga mälupöördus põhjustab lehekküljepöörduse vea

- Efektiivne juurdepääsuaeg (*Effective Access Time*)

$$EAT = (1 - p) \times ma + p \times pft$$

- ma = mälupöördusaeg
- pft = page fault'i käskude kulu + lehekülje väljasaalimise aeg
+ lehekülje sissesaalimise aeg + käsu taaskäivitamise aeg
- Domineerib lehekülje saalimise aeg
- Lehekülge pole vaja välja saalida, kui teda pole modifitseeritud!

Efektiivne juurdepääsuaeg — näide

- Olgu mälupöördusaeg $ma = 200\text{ ns}$
- Olgu *page fault*'i serveerimiseks kuluv aeg $pft = 25\text{ms}$
- Siis
$$EAT = (1 - p) \times 200 + p \times 25000000$$
$$= 200 + 24999800 \times p \text{ ns}$$
- Selleks, et EAT oleks mälupöördusajast mitte rohkem kui 10% aeglasem, peab $p \leq 1/1249990 \cong 8 \times 10^{-7}$

Protsesside loomine

Virtuaalmälu annab võimalusi efektiivsemaks protsesside loomiseks:

- *Copy-on-Write*
 - Vanemprotsess ja lapsprotsess jagavad leheküljetabeleid esialgu
 - Leheküljed on mõlema jaoks readonly, esimesel kirjutamisel tehakse kummalegi koopia
 - Efektiivsem protsesside loomine, eriti exec() tegijatele
- Aadressiruumi kaudu nähtavad failid (*Memory-mapped files*)
 - mmap() primitiiviga seatakse faili sisu kättesaadavaks mingis aadressivahemikus
 - Esimene pöördumine loeb andmed mällu, faili sulgemine kirjutab kettale, vahepeal võib kah kirjutada
 - read()/write() vs mmap()

Lehekülgede asendamine

- Mälu ülehõivamine (*overallocation, overcommiting*) — lubame protsessidele kokku rohkem mälu kui olemas on
- Kui reaalselt ka rohkem kasutatakse, tuleb lehekülgi asendada hakata
- Kasutame lehekülgede juures uut lippu *dirty* — märgib, et lehekülg on mälus modifitseeritud ja seega tuleb välja saalida enne vabastamist
- Viib lõpuni loogilise ja füüsilise mälu eraldamise — nüüd võivad nad sõltumatute suurustega olla

Lehekülje asendamisega protsess

- Otsime kettalt sissesaalimiseks vajaliku lehekülje
- Otsime vaba füüsilise mälu kaadri:
 - Kui leiame, kasutame seda
 - Kui ei leia, valime leheküljeasenduse algoritmiga ohvriks mingi lehekülje, mida välja saalida ning saalime välja
- Loeme vajaliku lehekülje (värskesse) vabasse kaadrisse
- Uuendame leheküljetabeleid
- Taaskäivitame protsessi

Leheküljeasenduse algoritmid

- Tahame madalaimat *page fault*'ide arvu
- Ette on antud kasutatavate kaadrite arv
- Algoritmide võrdlemiseks anname neile ette mingeid mälupöördusmustreid (näidisstringe) ja arvutame leheküljepöörduste vigade arvu
- Võime mäluaadresside pöördused grupeerida lehekülgede pöördusteks
- Üldiselt, mida rohkem kaadreid on kasutada, seda vähem leheküljepöörduste vigu tekib (aga mitte päris alati)

FIFO leheküljeasendusalgorithm (1)

- Asendame lehekülje, mis on kõige varem mällu toodud
- Näide:

1	2	3	1	2	4	1	4	2	3	2
1	1	1	1	1	4	4	4	4	3	3
	2	2	2	2	2	1	1	1	1	1
		3	3	3	3	3	3	2	2	2

- Lihtne realiseerida
- OS võib kergesti välja visata tihedalt kasutatava lehekülje

FIFO leheküljeasendusalgorithm (2)

- FIFO leheküljeasendusalgorithmi korral võib kaadrite arvu suurendamine viia leheküljepöördusvigade arvu suurenemiseni (nn. Belady anomaalia)
- Näide:

1	2	3	4	1	2	5	1	2	3	4	5
1	1	1	4	4	4	5	5	5	5	5	5
	2	2	2	1	1	1	1	1	3	3	3
		3	3	3	2	2	2	2	2	4	4

1	2	3	4	1	2	5	1	2	3	4	5
1	1	1	1	1	1	5	5	5	5	4	4
	2	2	2	2	2	2	1	1	1	1	5
		3	3	3	3	3	3	2	2	2	2
			4	4	4	4	4	4	3	3	3

Optimaalne algoritm (1)

- Asendame lehekülje, mida edaspidi kõige kauem ei kasutata
- Näide:

1	2	3	1	2	4	1	4	2	3	2
1	1	1	1	1	1	1	1	1	3	3
	2	2	2	2	2	2	2	2	2	2
		3	3	3	4	4	4	4	4	4

- Pole realiseeritav (kust me tulevikku ette teame?)
- Kasulik teiste algoritmide hindamiseks (saavutab teoreetilise maksimumi)

Optimaalne algoritm (2)

- Optimaalse algoritmi korral Belady anomaaliat ei esine
- Näide:

1	2	3	4	1	2	5	1	2	3	4	5
1	1	1	1	1	1	1	1	1	3	3	3
	2	2	2	2	2	2	2	2	2	4	4
		3	4	4	4	5	5	5	5	5	5

1	2	3	4	1	2	5	1	2	3	4	5
1	1	1	1	1	1	1	1	1	1	4	4
	2	2	2	2	2	2	2	2	2	2	2
		3	3	3	3	3	3	3	3	3	3
			4	4	4	5	5	5	5	5	5

LRU (*Least Recently Used*)

- Asendame kõige kauem mitte kasutatud lehekülje
- Näide:

1	2	3	1	2	4	1	4	2	3	2
1	1	1	1	1	1	1	1	1	3	3
	2	2	2	2	2	2	2	2	2	2
		3	3	3	4	4	4	4	4	4

- Eeldatakse, et minevik on hea tuleviku ennustaja
- Kui see eeldus pole täidetud, siis võib käituda väga halvasti
- LRU algoritmi korral Belady anomaaliat ei esine

LRU algoritm

- Näide:

1	2	3	4	1	2	5	1	2	3	4	5
1	1	1	4	4	4	5	5	5	3	3	3
	2	2	2	1	1	1	1	1	1	4	4
		3	3	3	2	2	2	2	2	2	5

1	2	3	4	1	2	5	1	2	3	4	5
1	1	1	1	1	1	1	1	1	1	1	5
	2	2	2	2	2	2	2	2	2	2	2
		3	3	3	3	5	5	5	5	4	4
			4	4	4	4	4	4	3	3	3

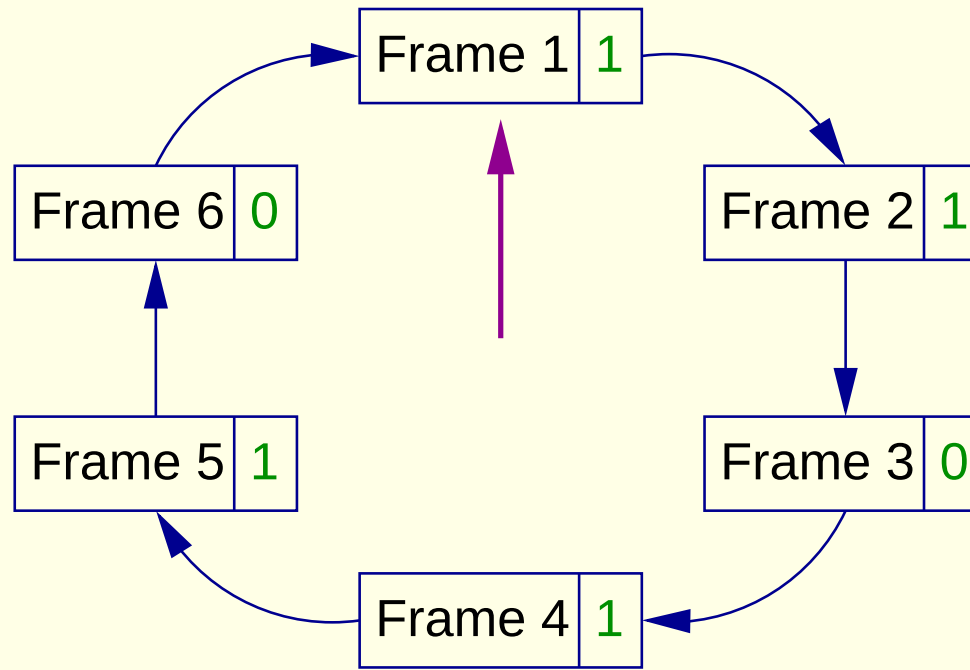
LRU realiseerimine

- Iga leheküljega seome mälupöördusaja loenduri (ajatempli)
 - Suurendatakse igal pöörusele sellele leheküljele
 - Ohvriks valitakse vähima loenduriga lehekülg
 - Kulukas realiseerida (vaja kontrollida kõik leheküljed)
- Haldame lehekülgede magasinini topeltseotud listina
 - Leheküljele pöördumisel tõstame selle magasinini tippu
 - Ohvriks valitakse magasinini põhjas olev lehekülg
 - Samuti kulukas realiseerida (nõuab 6-e viida muutmist)

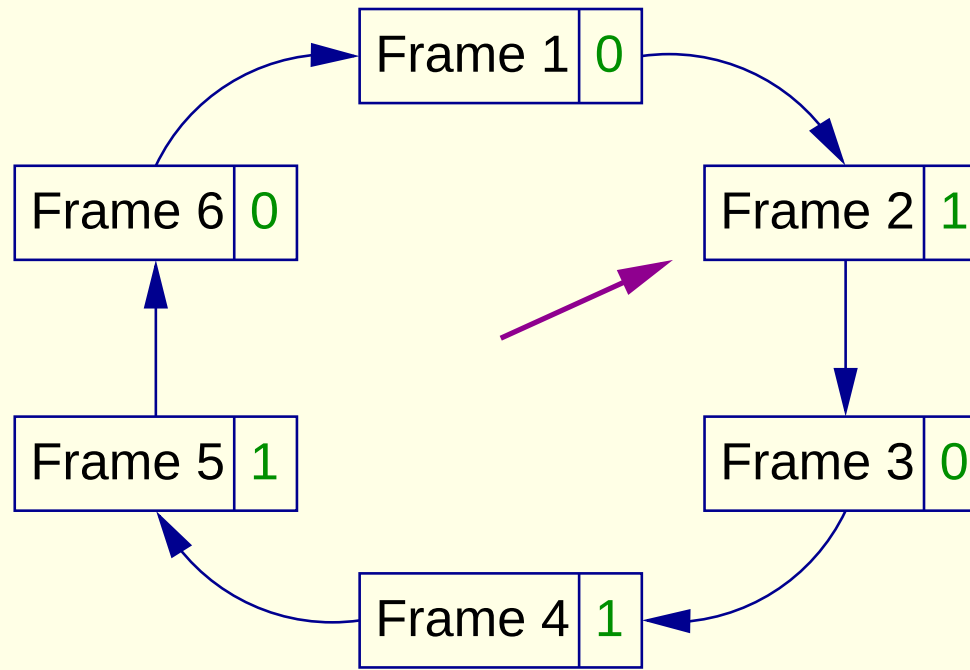
LRU lähendused

- Kasutuse bitt (*reference bit*)
 - Iga leheküljega seome lipu, algul 0
 - Lehekülje kasutamisel paneme lipu 1-ks
 - Ohvriks valime ühe nullise lipuga lehekülgedest (näiteks FIFO algoritmiga)
- Teine šanss (*second chance, clock*)
 - Vajame kasutuse bitti
 - Käime lehekülgi läbi tsükliliselt (nagu kellaosutiga)
 - Kui kasutuse bitt on 0, siis asendame lehekülje
 - Kui kasutuse bitt on 1, nullime ära, aga jätame lehekülje mällu
 - Ja nii tsükliliselt

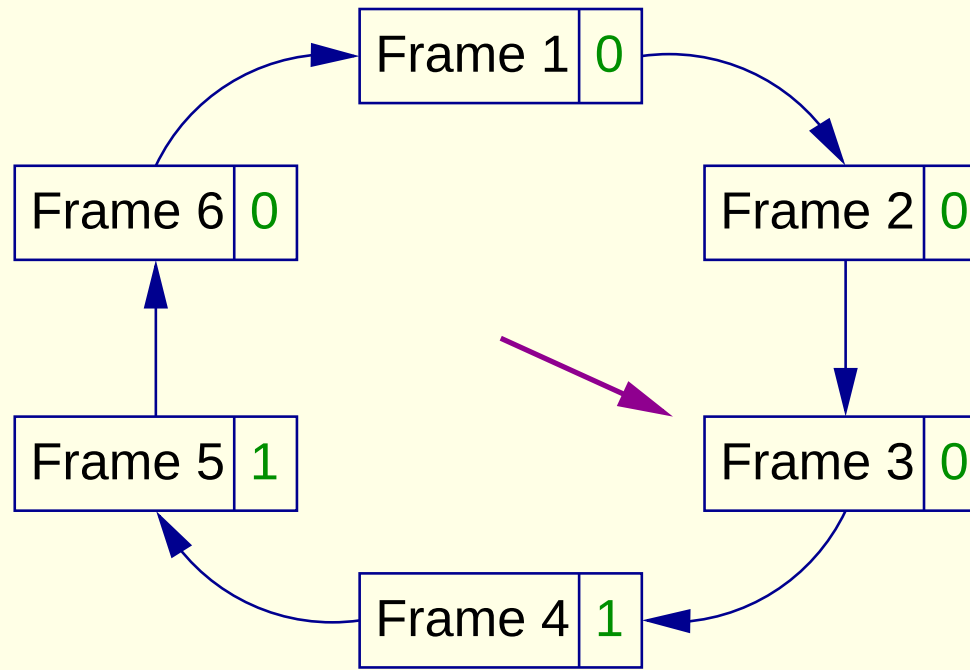
Teine šanss — näide



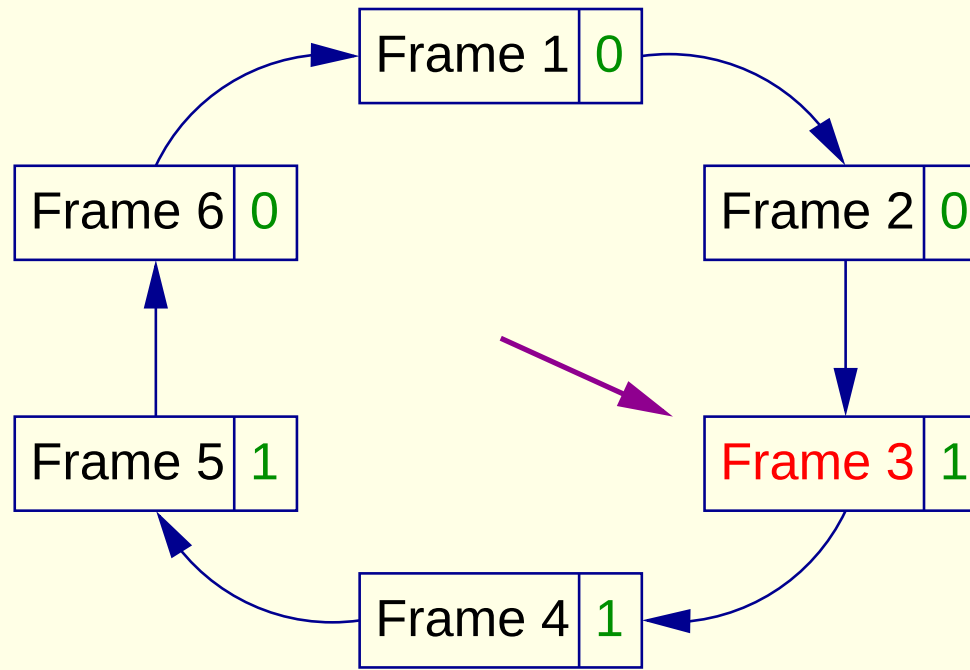
Teine šanss — näide



Teine šanss — näide



Teine šanss — näide



Loenduriga algoritmid

- Loendame lehekülje kaudu tehtud mälupöörduste arvu
- LFU (*Least Frequently Used*) — ohvriks valime vähima loenduri väärtusega lehekülje
- MFU (*Most Frequently Used*) — baseerub kaalutlusel, et vähim loenduri väärtus tähendab ilmselt, et lehekülg toodi just mällu ja on värske
- Kumbki pole eriti laialt kasutusel
- Leidub situatsioone, kus LFU on efektiivsem kui LRU (failisüsteemide cached näiteks)

Füüsilise mälu haldus

- Kuipalju ja missuguseid füüsilise mälu kaadreid eri protsessidele anda?
- Igale protsessil minimaalne ja maksimaalne lehekülgede arv
- Miinimum tuleb riistavara nõuetest — mitu lehekülge suudab üks instruktsioon puutuda (näiteks 6: 2 lehekülge käsu enda jaoks, 2 lähteadressi ja kaks sihtaadressi jaoks)
- Kaks põhilist hõivamise skeemi:
 - Fikseeritud lehekülgede arvud (võrdne või proportsionaalne)
 - Prioriteediga lehekülgehõivamine (suurema prioriteediga protsess võib väiksema prioriteediga protsessi lehekülgi välja tõrjuda)
- Globaalne asendus — kaadri võib võtta kogu mälust
- Lokaalne asendus — kaadrit saab võtta ainult oma mälust

Pekslemine (*trashing*)

- Kui protsessil pole piisavalt lehekülgi, on leheküljepöördusvigade sagedus väga suur
- Madal protsessorikasutus
- Opsüsteem leiab seetõttu, et võib protsesse juurde tuua, kuna protsessoriaega on
- Lisatakse veel protsesse, läheb veel aeglasemaks
- Pekslemine — protsess tegeleb peaaegu ainult saalimisega
- Lehekülgede saalimine töötab sellepärast, et korraga kasutatakse tegelikult ainult mingit kindlat hulka lehekülgi (töökomplekt — *working set*)
- Pekslemine tekib siis, kui see vajalik lehekülgede hulk ei mahu eraldatud kaadritesse ära

Muud (1)

- Lehekülgede eellaadimine (*prepaging*)
 - Protsessi käivitamisel või sisse saalimisel loetakse sisse terve töökomplekt
 - Sissesaalimisel tuuakse sisse ka ümberkaudseid lehekülgi
 - Kui leheküljed on kettal järjestikku, on mitme lehekülje korraga sisse lugemine palju efektiivsem, kui üksikhaaval
 - Võib laadida kasutuid lehekülgi, mis omakorda võib põhjustada vajalike lehekülgede eemaldamist

Muud (2)

- Lehekülje suuruse muutmine
 - Sisemine fragemnteerumine
 - Leheküljetabeli suurus
 - I/O kulu
 - Lokaalsus
- TLB ulatus (*TLB reach*) — mälu hulk, mille TLB suudab katta (TLB kirjete arv $\times$ lehekülje suurus)
- TLB ulatus võiks ideaalis ära katta töökomplekti
- Lisaks lehekülje suuruse muutmisele võib sisse tuua eri suurusega leheküljed ($\Rightarrow$ tarkvaraline TLB täitmine)

Muud (3)

- Programmeerimisel tuleks arvestada lokaalsusega
 - Korraga puutuda vähe mälu, lähestikku asuvaid mälualasid
 - Suuri mälutükke läbida järjestikku
- Lehekülgede lukustamine
 - I/O jaoks
 - Äsja sisse saalitud lehekülgede lukustamine
 - Reaalajaprotsesside jaoks
 - Kasutja poolt juhitud lukustamine on reeglina privilegeeritud operatsioon