**1. Segmentation (Сегментация) — логикалық бөлу**

Не істейді: Бағдарламаның жадын логикалық мағынасы бойынша бөледі: мысалы, код бөлігі, мәліметтер бөлігі, стек бөлігі, т.б. Әр бөлік – сегмент деп аталады.

* Код сегменті (Code segment)
* Мәліметтер сегменті (Data segment)
* Стек сегменті (Stack segment)
* Heap сегменті және т.б.

Мысал:  
Егер бағдарламада 3 бөлік болса:

* Код – 50 KB
* Мәліметтер – 30 KB
* Стек – 20 KB

Онда жадта 3 бөлек сегмент бөлінеді. Олар физикалық жадыда әртүрлі жерлерде орналасуы мүмкін.

Қайда қолданылады:

* Бағдарламалау логикасын нақты бөліктерге бөлгіңіз келсе.
* Әр бөлікке бөлек қорғау және құқық орнатқыңыз келсе.
* Кейде компиляторлар мен ОС логикалық тұрғыда бөлімдерді бөлу үшін пайдаланады.

## **2. Paging (Пейджинг) — физикалық бөлу**

Не істейді: Бағдарламаның жадын тұрақты бірдей өлшемді беттерге (pages) бөледі. Физикалық жады да сол өлшемде кадрларға (frames) бөлінеді.

* Әр бет (page) әдетте 4 KB немесе 8 KB болады.
* Виртуалды беттер физикалық кадрларға тәуелсіз ретпен орналаса алады.

Мысал: Бағдарлама 12 KB жад сұрады → 3 бетке бөлінеді (4 KB × 3).  
 Физикалық жадыда: бет 1 → frame 7, бет 2 → frame 2, бет 3 → frame 10 — яғни олар қатар емес, әр жерде болуы мүмкін.

Қайда қолданылады:

* Жадыны үздіксіз емес, шашыраңқы түрде тиімді пайдаланғыңыз келсе.
* External fragmentation-нан құтылғыңыз келсе.
* Көптеген қазіргі операциялық жүйелерде (Windows, Linux, macOS) paging — негізгі механизм.

Қысқа салыстыру (қазір тек түсінік деңгейінде)

|  |  |  |
| --- | --- | --- |
| Параметр | Segmentation | Paging |
| Бөлу тәсілі | Логикалық бөлу (код, стек, т.б.) | Тұрақты өлшемді беттерге бөлу |
| Блок өлшемі | Өзгермелі | Тұрақты (мыс. 4 KB) |
| Басым қолдану | Бағдарламаның логикалық құрылымын көрсету | Жадыны тиімді басқару |
| Фрагментация | External fragmentation болуы мүмкін | Internal fragmentation болуы мүмкін |
| Қорғау | Әр сегментке бөлек | Әр бетке бірдей |
| Мысал | Код сегменті 50 KB | 12 KB → 3 бет |

**Қорытынды:**

* Segmentation – логикалық тұрғыда ыңғайлы, бағдарлама бөліктерін бөлек қарауға көмектеседі.
* Paging – физикалық тұрғыда тиімді, жадты жақсырақ пайдаланады.

Ал ең заманауи жүйелерде бұл екеуі бірге қолданылады:

* Алдымен бағдарлама сегменттерге бөлінеді,
* Әр сегмент өз ішінде беттерге бөлінеді.  
   Мұны «segmented paging» деп атайды – бұл әдіс логикалық құрылым мен физикалық тиімділікті біріктіреді.

**3. Виртуалды жад жүйесінде жылжу (thrashing)**

Виртуалды жад жүйесінде жылжу (thrashing) — бұл процесс компьютердің жадын тым көп ауыстырып, жүйенің негізгі жұмысына уақыт бөлмей, жадқа үнемі кірумен және деректерді ауыстырумен айналысып қалуы.

Яғни, процесс жадқа қажетті деректерді табу үшін үнемі диск пен жад арасында деректерді көшіріп отырады, соның салдарынан компьютер баяу жұмыс істейді.

Қалай анықтауға болады?

* Компьютер баяулайды.
* Процестердің орындалуы өте ұзаққа созылады.
* Жадқа жиі кіру және дискке үнемі жазу/оқу байқалады.

Қалай жоюға болады?

* Қосымша бағдарламаларды жауып, жадты босату.
* Компьютерге көбірек физикалық жад (RAM) орнату.
* Виртуалды жадтың көлемін дұрыс баптау.
* Процестердің санын азайту, яғни бір уақытта көп бағдарлама ашпау.

Қысқаша айтқанда, жылжу — бұл жадтың тым жиі ауысуынан жүйенің баяулап кетуі. Оны шешу үшін жадты тиімді пайдалану және жүйені оңтайландыру қажет.

**4. Виртуалды және физикалық адрестер мысалы**

Түсінік:

Компьютердегі әр процесс өз «виртуалды адрестерін» қолданады. Бұл — жадтың логикалық бөлігі. Ал нақты (физикалық) адрестер — бұл компьютердің жадысының нақты ұяшықтары.

Мысал:

• Бет (page) өлшемі – 4 KB (4096 байт)

• Виртуалды адрестер: 0 – 16 KB

Яғни, 4 бет бар:

• Бет 0 → 0–4 KB

• Бет 1 → 4–8 KB

• Бет 2 → 8–12 KB

• Бет 3 → 12–16 KB

**5. Page Replacement алгоритмдері**

Виртуалды жадта кей беттер дискте сақталады, ал тек кейбіреуі оперативті жадта (RAM). Егер жаңа бет қажет болса, біреуін шығарып, жаңасын қою керек. Осы кезде бет алмастыру (page replacement) алгоритмдері қолданылады.

Қысқа мысал:

Беттер реттілігі: 1, 2, 3, 1, 4, 5

Жадыда 3 орын бар.

• FIFO → 3 бет қате (page fault) көп болады.

• LRU → жиі қолданылған беттерді сақтап қалады.

• Optimal → ең аз бет қате береді.

• Clock → тиімді әрі жеңіл орындалады.

**6. Belady’s anomaly (Белади феномені)**

Түсінік:

Кей жағдайларда жад көлемін көбейткенде, бет қатесі (page fault) азаймай, керісінше көбейеді.

Бұл – Белади феномені деп аталады.

Мысал (FIFO алгоритмінде):

Беттер тізбегі: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

• Егер жадыда 3 бет болса → 9 page fault

• Егер жадыда 4 бет болса → 10 page fault

Яғни, жады артса да, қателер көбейді — бұл Belady’s anomaly.

⸻

**7. TLB (Translation Lookaside Buffer) рөлі**

TLB – бұл арнайы жылдам кэш жад, ол виртуалды → физикалық адрестердің сәйкестігін сақтайды.

Рөлі:

• Адрестерді тез аударады

• Процесті баяулатпайды

• Егер TLB-да адрес табылмаса → бет кестесінен іздейді

Қысқаша:

TLB → жады адрестерін жылдам табуға көмектесетін «mini-карта» сияқты.

**8. Жады фрагментациясы**

Жады фрагментациясы — бұл компьютер жадында (RAM немесе басқа сақтау құрылғыларында) орын алатын құбылыс, онда бос орындар кішігірім және бөлшектенген бөліктерге бөлініп, үлкен көлемдегі үздіксіз бос орындар болмай қалады. Нәтижесінде, жадты тиімді пайдалану қиынға соғады.

Фрагментация екіге бөлінеді:

1. Ішкі фрагментация — бөлінген жады блоктары үлкенірек болып, пайдаланылмаған бөлігі болады. Мысалы, егер сізге 10 байт қажет болса, бірақ 16 байт бөлінсе, қалған 6 байт бос қалады.
2. Сыртқы фрагментация — бос орындар жадта шашыраңқы орналасып, қажетті көлемдегі үздіксіз жады бөлігі табылмайды, сондықтан үлкенірек блоктарды бөлу мүмкін болмай қалады.

Жады фрагментациясы жүйенің жұмысын баяулатады және жадты тиімді қолдануға кедергі болады.

Қысқаша айтқанда: жады фрагментациясы — бұл жадтағы бос орындардың бөлшектеніп, үздіксіз үлкен бөлік ретінде болмай қалуы.

**9. Page fault кезінде не болады**

Page fault – процесс қажет бетті RAM-да таппаған жағдай.

Қадамдар:

1. Процесс бетті сұрайды.

2. ОЖ (операциялық жүйе) қарайды — бет RAM-да жоқ.

3. Page fault пайда болады.

4. ОЖ дискке жүгінеді, қажетті бетті табады.

5. Егер RAM толы болса — бір бетті шығарып тастайды (page replacement).

6. Жаңа бетті RAM-ға жүктейді.

7. Бет кестесін жаңартады.

8. Процесс жалғасады.

**10 сұрақ:**

1. Бөлінуші (contiguous) жад бөлу — First Fit, Best Fit, Worst Fit

Қысқаша: бұл алгоритмдер физикалық (немесе логикалық) жадта үздіксіз блоктарды іздеп, процесстің сұраған көлемін сол блокқа орналастырады.

First Fit (бірінші қолайлы): бос блоктарды солдан оңға қарай тексеріп, бірінші рет жеткілікті үлкен блок тапса — сол блоктан бөледі.

Артықшылығы: жылдам (іздеу қысқа ортада).

Кемшілігі: уақыт өте келе бастың (beginning) жады көп фрагменттеледі (external fragmentation).

Best Fit (ең жақсы): барлық бос блоктарды қарап, сұранысқа ең жақын — ең аз артық мөлшер қалдыратын блокты таңдайды.

Артықшылығы: бастапқы қалдық азайтуға тырысады (теоретикалық түрде).

Кемшілігі: бос блоктарды іздеу көп уақыт алады; кішкентай қалдықтар көп пайда болып, фрагментация проблемасы артуы мүмкін.

Worst Fit (ең жаман): ең үлкен бос блокты таңдап, оған орналастырады (мақсат — үлкен блокты бөлу арқылы кейінгі үлкен сұраныстарға орын қалдыру).

Артықшылығы: үлкен сұраныстарға орын қалуы мүмкін.

Кемшілігі: нақты тәжірибеде жиі тиімсіз.

Нақты мысал (кіші): бастапқы бос блоктар: [100, 500, 200, 300, 600] (байт деп есептейік). Процесстер сұрамалары: [212, 417, 112, 426].

First Fit нәтижесі:

212 → 500-блокқа (қалды: 288),

417 → 600-блокқа (қалды: 183),

112 → сол қалған 288-блоктан (қалды: 176),

426 → орналастырылмайды (басқа блоктар жеткіліксіз).

(нәтиже алым-қалдық: [100,176,200,300,183])

Best Fit нәтижесі:

212 → 300-блокқа (қалды: 88),

417 → 500-блокқа (қалды: 83),

112 → 200-блокқа (қалды: 88),

426 → 600-блокқа (қалды: 174).

(қалды: [100,83,88,88,174])

Worst Fit нәтижесі:

212 → 600-блокқа (қалды: 388),

417 → 500-блокқа (қалды: 83),

112 → қалған 388-блокқа (қалды: 276),

426 → орналастырылмайды.

(қалды: [100,83,200,300,276])

Қорыту: нақты жүйеде әдетте First Fit немесе Buddy жүйелері қолданылады; Best/Worst практикалық қиындықтар береді.

## 2. Жадты басқару әдістері (Memory Management Techniques)

Segmentation (сегментация) мен Paging (пейджинг) — бұл операциялық жүйеде қолданылатын виртуалды жадыны басқарудың екі негізгі механизмі.  
 Олар – виртуалды адрестерді физикалық жадқа сәйкестендірудің екі түрлі әдісі.

Мақсаты:

* бағдарламалардың жадқа тиімді және қауіпсіз орналастырылуын қамтамасыз ету,
* процестер бір-бірінің жадына кіріп кетпеуін сақтау,
* физикалық жадты үнемді пайдалану,
* бағдарламаның логикалық құрылымын жадқа ыңғайлы етіп орналастыру.

Неліктен керек?

## Біздің бағдарламалар логикалық тұрғыда үлкен жадты қолданғысы келеді, бірақ физикалық жад шектеулі және фрагменттелген болуы мүмкін. Мысалы:

* Бағдарлама 100 МБ жад сұрайды, ал RAM-де 4×25 МБ бос орын бар. Үздіксіз 100 МБ жоқ, бірақ 4 бөлек жерде бар.
* Егер тек үздіксіз бөлу болса — бағдарлама жүктелмейді.
* Ал segmentation немесе paging қолдансақ — ол 100 МБ логикалық жадты бөліктерге бөліп, физикалық жадының әр жеріне орналастыра алады.



