

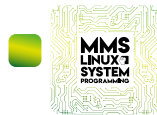
Системно програмиране за Линукс

Управление на паметта. Виртуална памет и странична организация.

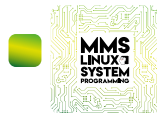
Ангел Чолаков



21.04.2021г.



This work is licensed under a Creative Commons
“Attribution-ShareAlike 4.0 International” license.



Съдържание I

- 1 Въведение
- 2 Особенности на микропроцесорната архитектура
- 3 Адресно пространство
- 4 Същност на виртуализацията
- 5 Сегментация
- 6 Апаратно осигуряване за виртуализация на паметта
- 7 Странично-сегментна организация
- 8 Примери при i386 и x86_64
- 9 ОС в ролята на диспечер на паметта
- 10 Управление на виртуалната памет в Линукс
- 11 Практически пример
- 12 Заключение



Цел на презентацията

■ Да опита да поясни:

- какво е **адресното пространство**, в което се зареждат и изпълняват заданията;
- какви са основните принципи за управление и разпределяне паметта;
- каква е концепцията на **страничната организация**;
- какви основни механизми за разпределяне на паметта съществуват;
- каква е ролята на **мениджъра на паметта (MMU)** в една система



Роля на ОС при управлението на паметта

■ Ангажименти:

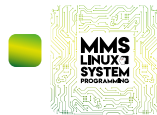
- подsigури средства за управлението на физическата памет;
- освободи потребителските процеси от отговорността сами да се грижат за физическата организация на паметта;
- гарантира, че всеки процес ще се изпълнява в обособена и изолирана област от адресното пространство без възможност за директен достъп до паметта, предоставена на други процеси;
- изгради абстрактен модел на логическа организация на достъпваната памет (виртуализация);
- менажира разпределението и степента на заетост на физическата памет



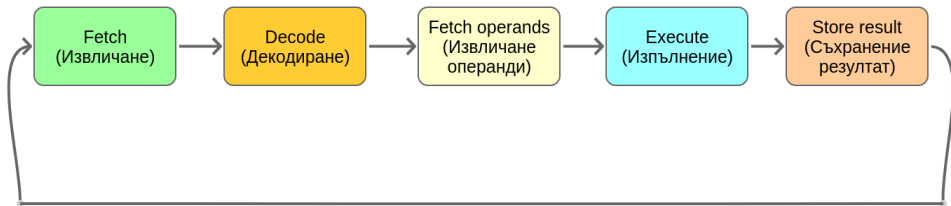
Влияние на микропроцесорната архитектура

■ Фактори, с които ОС се съобразява:

- какъв **модел на изпълнение на програмни инструкции** е реализиран;
- какъв набор от регистрови структури и операции за адресиране и достъп до паметта са налични;
- какви **механизми за междинно буфериране на региони** от основната памет (кеширане) са реализирани и поддържани;
- какви **апаратни възможности за изолация и защита на адресното пространство** архитектурата предоставя



Илюстрация на инструкционен конвейер



pic. based on work by HydenB, CC BY-SA 4.0 via Wikimedia Commons



Конвейер на инструкциите

■ Разяснение на фазите:

- **извличане** - прочитане на поредна инструкция от адрес, съхранен в програмния брояч и запазването и в регистъра на изпълнимата инструкция за изпълнение;
- **декодиране** - интерпретиране на инструкцията от блок декодер. При достъп до паметта се определя и ефективният адрес, като изчислението е зависимо от апаратния модел на сегментиране на паметта;
- **изпълнение** - изпълнение на извлечената инструкция, включващо прочитане на необходимите операнди, определяне на кода на операцията и математическа или логическа обработка от АЛУ, последвано от съхранение на резултата и установяване на флагове на състоянието;



Физически структури, съхраняващи данни

■ Обособяват се:

- **регистри на процесора:** със специално/общо предназначение, даннови, адресни, буферни и др;
- **памет:** бърза асоциативна (**кеш памет**) и **оперативна** - с адресни и даннови шини за управление;
- **вторична дискова или флаш памет:** за дългосрочно съхранение на данни и програми



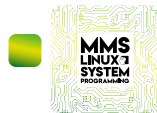
Какво е адресно пространство?

■ Дефиниция:

- краен диапазон от адреси, ограничен от изчислителния хардуер, в който процесите се зареждат и изпълняват;

■ Видове:

- **физическо** - множество от реални физически адреси, определено от размера на привързаната оперативна памет;
- **логическо** - виртуално множество от логически адреси, с които зарежданите и активни задания оперират, като този набор е представен по различен начин от действителната физическа организация



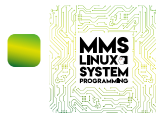
Защо е необходимо това разграничение?

Позволява на ОС да:

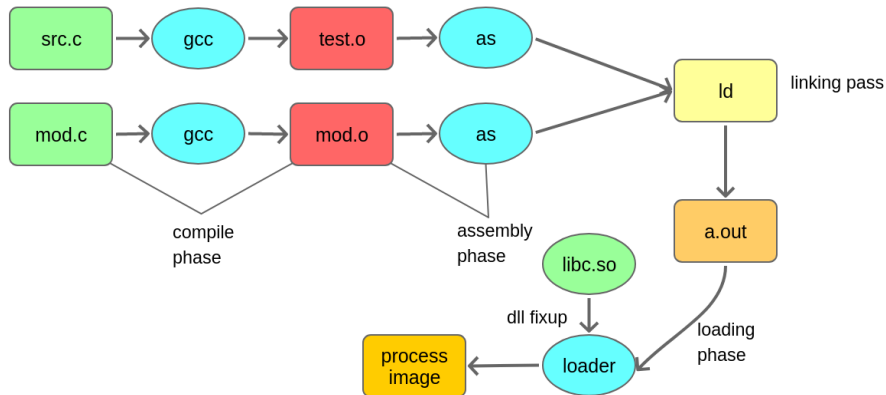
- позволява на ОС да поеме контрол върху разпределението физическите ресурси;
- прави възможно ОС да осигури изолация на адресното пространство на процесите и защита на данните, с които те оперират;
- освобождава разработчиците от необходимостта да се грижат за това как приложенията се поместват в паметта и оперират с нея;
- позволява изпълнението на множество програми, чиито изисквания могат да надвишават обема на наличната памет;



pic. by AliWijaya, CC BY-SA 3.0



Да си припомним етапите по създаване на изпълним файл



pic. based on <https://en.wikipedia.org/wiki/Compiler>



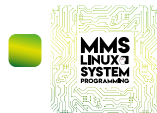
Обособяват се три основни фази

■ Пояснение:

- **компилиране** - при генериране на обектна програма компилаторът изобразява символичните адреси в подходяща форма - преместваема най-често;
- **асемблиране и свързване** - обединяване на различни обектни фрагменти (подпрограми) и разрешаване на референции към външни библиотечни функции;
- **зареждане** - зареждане на процеса в паметта и представяне на преместваемите (relocatable) адреси в абсолютни



Как да опростим описаните етапи, така че ОС да поеме контрол върху паметта? Какви базови политики за разпределение на паметта за всеки изпълняван процес съществуват?



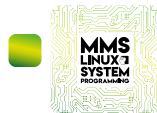
Статично разпределение на паметта

■ Особенности:

- **статичното разпределение:** налага предварително изчисляване на размера памет, необходим за разполагане и изпълнение на дадена задача;
- цялата нужна оперативна памет се заделя наведнъж и след настройка на физическите адреси от свързващия редактор или зареждащата програма, изображението на процеса в паметта е фиксирано



Бихте ли посочили примерни недостатъци на този подход?



Динамично разпределение на паметта

■ Особенности:

- **динамичното разпределение:** разчита на процедури по динамично заделяне на региони от паметта по време на изпълнение на процеса;
- работата на компилатора и свързващия редактор се облекчава, като се използват фиктивни логически адреси и отмествания, които се преобразуват във физически непосредствено преди всяко обръщение от паметта впоследствие;
- присвоените области реална памет подлежат на динамично преместване, свиване или разширяване - също в хода на работа



Как се подсигурява виртуализацията?

■ Посредством апаратна поддръжка на:

- **регистрови и таблични структури** за трансляция между физически и логически адреси;
- **странично-сегментна организация** на виртуалното адресно пространство;
- механизми, подпомагащи въвеждането, извеждането и размяната на блокове (страници) между основната и вторичната памет



Виртуализация и техники за адресно транслиране



pic. based on https://en.wikipedia.org/wiki/Virtual_memory



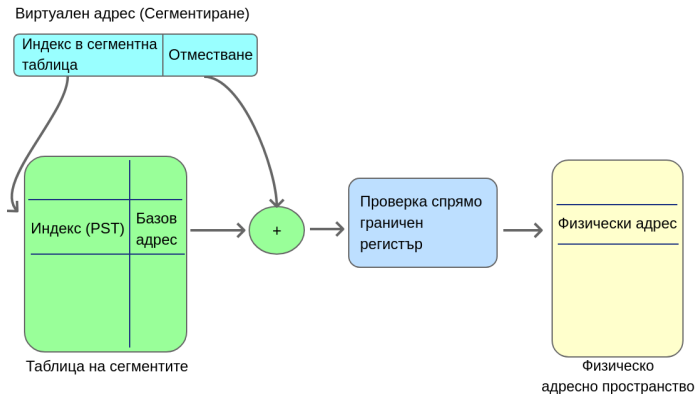
Сегментация с базови и гранични регистри

■ Същност:

- за всяка задача се привързва **таблица със сегментите**;
- всеки нейн запис е структура с два елемента - **базов адрес** и **дължина на региона** (граница);
- записите на сегментната таблица се съхраняват и модифицират посредством двойка регистри: **базов и граничен**;
- по време на изпълнение виртуалните адреси се превръщат във физически динамично с помощта на привързаната сегментна таблица;
- към прочетения базов адрес за даден индекс се прибавя отместването и се прави проверка дали границата е надвишена



Сегментация: диаграма на адресиране



pic. based on <https://pages.cs.wisc.edu/~remzi/OSTEP/vm-segmentation.pdf>



Характеристики на сегментацията

■ Позволява:

- позволява **динамична релокация** на разделите на изпълнимия процес по време на изпълнение;
- нов начален адрес на сегмент може да бъде указан чрез промяна на стойността на базовия регистър;
- респективно нов размер би могъл да бъде настроен с помощта на граничния регистър
- заданията могат да се разполагат в паметта на произволно място и паметта да се уплътнява;



Пример при сегментно разпределение

■ Нека да предположим, че:

- началото на heap региона започва от виртуален адрес 4096 (4KB);
- съдържанието на базовия регистър за heap е 48 KB във физическата памет;
- горната граница е настроена на 64 KB

■ На какъв действителен адрес ще кореспондира виртуален адрес 4100?

- ако просто добавим 4100 към стойността в базовия регистър, няма да получим коректен резултат (52100);
- първо трябва да определим отместването спрямо указаното начало, което в случая е: 4096, като получаваме $4100 - 4096 = 4$
- вземаме изчисленото отместване и го добавяме към съдържанието на базовия регистър, за да получим верния резултат: 48004



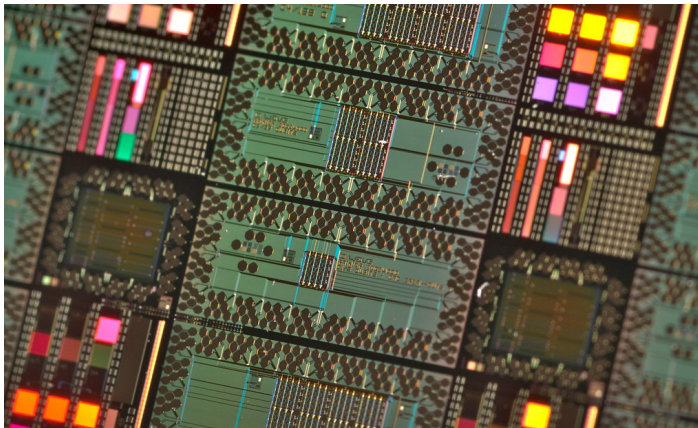
Недостатъци на сегментацията

■ Можем да споменем някои:

- при динамична честа промяна на границите на сегментите се получава отчетлива фрагментация на физическата памет;
- зареждащата програма има задачата да намери свободна памет за всяко помествано задание, е необходима реорганизация на вече използваните раздели, което е невинаги е оптимално;
- ако няма достатъчно свободна физическа памет за постъпване на нов процес, той ще бъде отложен неопределено;
- друг много съществен недостатък е неимоверното нарастване на сегментните таблици при увеличаване на размера на паметта



Кои са CPU структурите, ангажирани с паметта?



pic. by jurvetson, CC BY 2.0 via creativecommons.org



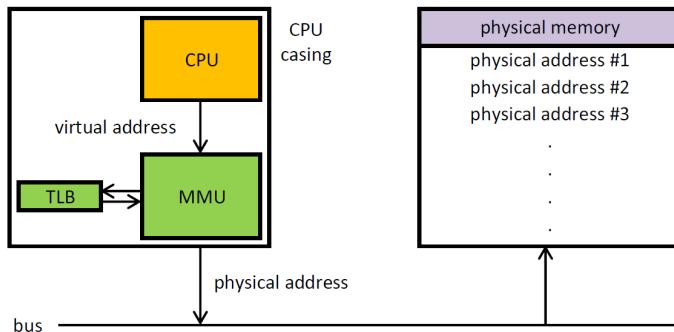
Виртуализация: апаратно осигуряване

■ Нужно е CPU да разполага с:

- специализиран **модул за арбитражиране (MMU)**, снабден с поне двойка регистри: базов и граничен;
- набор от **привилегировани инструкции** за достъп/заявка до услугите на MMU;
- привилегировани инструкции за **регистриране и обработка на изключителни ситуации**;
- способност за **генериране на събития**, сигнализиращи неправилен или непозволен достъп до паметта и/или защитените регистри;
- **прекъсване на изпълнявания процес** при детекция на събитие, маркиращо изключителна ситуация



Диаграма на MMU

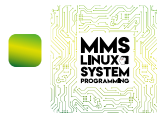


CPU: Central Processing Unit

MMU: Memory Management Unit

TLB: Translation lookaside buffer

pic. by Mdjango, Andrew S. Tanenbaum, CC BY-SA 3.0 via Wikimedia Commons



MMU: еволюционно развитие

■ Част от времевата линия на еволюция:

- Burroughs B5000 (1961) - една от първите компютрни системи (след Atlas), реализиращи идеята за виртуална памет макар и без обособяването на MMU;
- IBM System/360 Model 67 (1965) - разчита на отделен хардуерен модул извън системния процесор, който се грижи за транслиране на адресите;
- Sun-1 - first generation Unix SBC (1982) - пример за едно от първите решения със странично-сегментна организация, включващо и обработка на флагове за защита на данните;
- Intel i386CXSБ (1989) - интегрира MMU в структурата на микропроцесора и разполага с множество възможности за конфигурация;
- повечето съвременни микропроцесори разполагат с вграден мениджър на паметта (MMU)



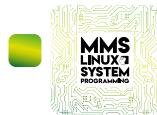
Сегментация: роля на ОС

■ Отговорности:

- поддържане на допълнителни управляващи структури за всеки процес, описващи атрибути на привързаните сегменти (**PCB**);
- създаване и обслужване на списъци с тези управляващи структури, чрез които паметта се уплътнява оптимално;
- механизми за поемане на контрол при обработване на отчетена изключителна ситуация от процесора (**exception handlers**)



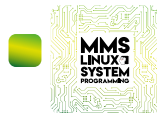
**Удачно ли е MMU да пази винаги регистрови записи или сегменти
таблицы?**



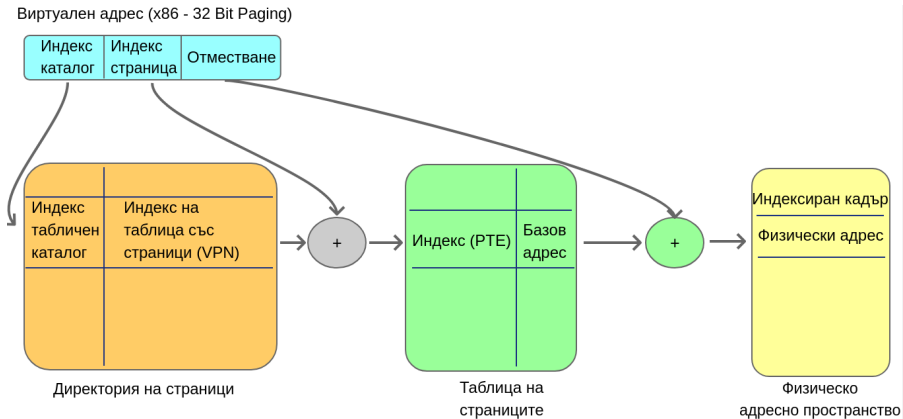
Виртуализация: странично-сегментна организация

■ Същност:

- вместо сегменти с начало и променлива дължина виртуалното адресно пространство се дели на малки порции с фиксиран размер, наречени **страници**;
- физическата памет се третира като линеен масив от слотове също с определен размер, наречени **кадри**;
- адресното транслиране става с помощта на специализирани **таблици на съответствие (PT)**, генерирани и привързани към управляващите структури на всеки процес;
- страници от виртуалното адресно пространство могат динамично да се въвеждат и извеждат от основната памет и съхраняват върху привързан носител



Диаграма на странично-сегментна организация



pic. based on <https://pages.cs.wisc.edu/~remzi/OSTEP/vm-paging.pdf>



Механизъм на адресно транслиране

■ Стъпки:

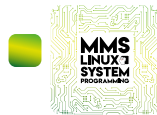
- най-простичката възможна форма е **схема за адресиране с два каталога: директория на таблици** (по една за всеки процес) и реферирана **таблица на страниците**;
- ОС индексира директорията посредством **виртуален индекс - (номер) на таблица**, след което локализира запис в таблицата, представен като част от **линейния виртуален адрес**;
- прочитането на запис в таблицата представя базов физически адрес на съответствие, насочен към номер на физически кадър;
- финалният физически адрес се получава чрез прибавяне на отместване



Защо точно механизъм с междинни каталози?

■ Стъпки:

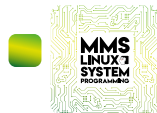
- подходът позволява физическата памет да се раздели на по-малки фрагменти с фиксирана дължина, наречени кадри;
- всеки процес получава референции към тези кадри посредством привъзразни таблица на сегментите и таблица на страниците;
- разделянето на обръщението на две или повече нива позволява по-гъвкаво адресиране на големи обеми памет без това да води до прекомерно нарастване на размера, ако се използва единична таблица;
- всеки елемент в таблицата на страниците не кореспондира към адрес на дума в паметта, а към регион (кадър) с удачно избрана размерност (4KB), така че да се намали общият брой на референциите без това да доведе обаче до по-съществена фрагментация



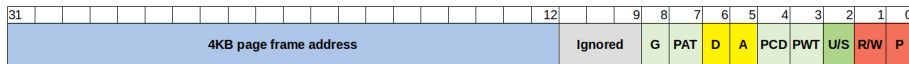
Странично-сегментна организация: MMU

■ Различия:

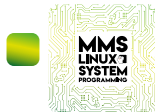
- таблиците със съответствия обикновено се съхраняват също в паметта и могат да са част от виртуалното адресно пространство;
- MMU се разтоварва от ангажимента да пази големи обеми данни, а обикновено подпомага процедурата по адресна трансляция, като ползва междинни каталожни данни от паметта;
- спомагателен блок от процесора (**TLB: Translation Lookaside Buffer**) служи за кеширане на вече достъпвани и известни таблични референции



Формат на линейно адресиране при i386



pic. based on <https://pages.cs.wisc.edu/~remzi/OSTEP/vm-paging.pdf>



Линейно адресиране при i386

■ Размерност на полето за адресиране на страниците:

- 20 бита разредност за виртуален номер на страница, което прави таблица с 2^{20} максимално възможни записа за всеки процес;
- ако всеки запис в таблицата реферира дума от по 4 байта и съдържа адрес също с дължина 4 байта, това прави по 4MB таблична структура, необходима за всеки процес в системата, което е неприемливо;

■ Подход за оптимизация:

- за да се редуцира паметта за табличните структури, необходими за обръщение към сегменти от изпълнимия файл със същия размер от 4MB, записите в каталога и таблицата на страниците се редуцират до 1024;
- 1 запис в сегментната таблица сочи към таблица с 1024 странични референции, всяка от които сочи към регион с размерност 4KB;
- така за сегмент с дължина, по-малка или равна на 4MB, е необходим само един запис в каталога и съответно една таблица на страниците;
- **размерът на всяка от двете таблици е по 4KB ($1024 * 4B$) вместо една таблица, заемаща 4MB;**
- допълнителни битови полета са заделени за флагове за проверка на достъпа и нанесена промяна на реферираните данни



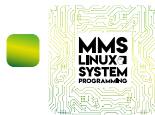
Линейно адресиране при i386

■ Описание на полетата:

- **PFN** номер на страничен кадър;
- **P** - указва дали страницата е заредена;
- **R/W** - показва дали е позволено четене/запис;
- **U/S** - индикация за режим на работа: потребителски/привилегирован;
- **G, PWD, PCD, PAT** - свързани с избор на режим за хардуерно кеширане;
- **A** - индикация за направен достъп;
- **D** - индикация за направена модификация



А колко записа в сегментната таблица на едно ниво биха били необходими за представяне на един 4МВ изпълним сегмент?



Какво е локалност на данни и адреси?

■ Дефиниция:

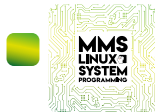
- **пространствената локалност** описва тенденция изпълнението на поредна инструкция да е обвързано с прочитане на данни от адрес, близък на предходния поради предимно линейния последователен характер на природата на изпълнение на някои функции;
- **времената локалност** се свързва с тенденцията за отложен повторен достъп до вече реферирани адреси при обработка на циклични операции например



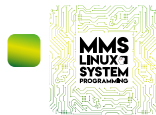
Роля на MMU и TLB

■ Задачи:

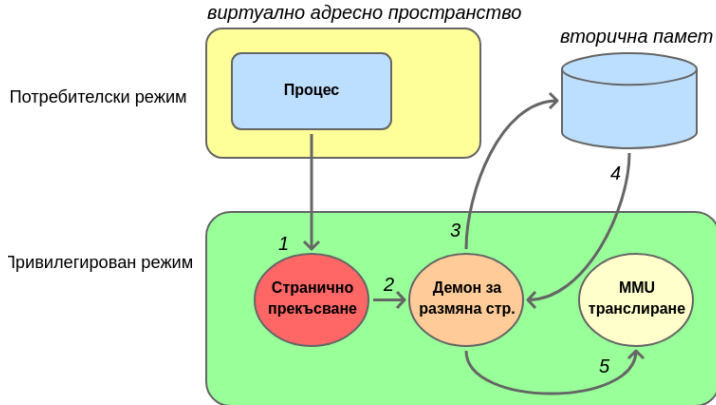
- да се възползват оптимално от **пространствената** и **времева** локалност на реферираните от един процес адреси;
- да подсиgurят механизми за буфериране (кеширане) на често достъпвани елементи;
- да осигурят апаратни възможности за контрол на достъпа и обработка на странични прекъсвания



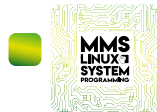
Как ОС диспечерира паметта?



Механизъм на въвеждане и извеждане на страници



pic. based on https://en.wikipedia.org/wiki/Page_table



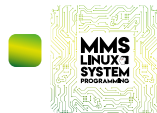
Механизъм на въвеждане и извеждане на страници

■ Същност:

- ОС поддържа вътрешни списъчни структури от данни, чрез които проследява честота на употреба на въведените страници;
- рядко достъпвани и ползвани страници периодично се извеждат от основната във вторичната памет;
- при заявка за достъп MMU обслужва събитие по **странично прекъсване**, което зарежда търсената страница обратно в основната памет;
- когато процес направи неправилен достъп до недостъпен виртуален адрес, ОС генерира **прекъсване за грешка (SIGSEGV)** и преустановява работата на процеса



А как виртуалната памет се представя в Линукс при x86_64?

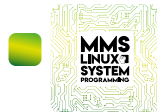


Карта на виртуалната памет в Линукс при x86_64

Примерна карта на виртуална памет в
Линукс - x86-64 (KPTI)



pic. based on https://www.kernel.org/doc/Documentation/x86/x86_64/mm.txt



x86_64: режим с адресна изолация

■ Особенности:

- фиксиран регион е резервиран само за код и данни на ядрото;
- ядрото традиционно се разполага в най-горните виртуални адреси;
- текстовата секция на ядрото обикновено се привързва към физически адрес 0;
- част от виртуалното адресно пространство на ядрото е резервирано за споделена употреба от всички процеси в системата



Управление на виртуалната памет в Линукс

■ Архитектура:

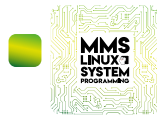
- традиционно се ползват таблици на три (или повече) нива;
- първото ниво е **глобална директория на страниците**, чиито записи сочат към междинни справочници;
- всеки **междинен каталог** пази адреси на таблици от последващо ниво до достигане на привързана таблица на страниците;
- таблицата на страниците съдържа показалци към физически адреси на страничните кадри



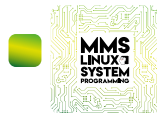
Управление на паметта в Линукс: kswapd

■ Механизми:

- разчита на **платформена поддръжка за представяне на състоянието на всяка страница** чрез две битови полета в PTE: **Accessed** (достъп) и **Dirty** (модификация) бит;
- битът за достъп се установява при наличие на операция за четене, писане или изпълнение;
- битът за модификация е индикация за извършена промяна в съдържанието на страницата;
- специализиран демон на ядрото **kswapd** периодически инспектира битовите полета и нулира битовете за достъп;
- при недостиг на системна памет, kswapd изхвърля страници от първичната памет и ги пренася във вторичната;
- стари и недостъпвани страници се изхвърлят от паметта приоритетно



Как Линукс оптимизира процеса по изграждане на старнични таблици, когато процес създава свое копие?



Механизъм copy-on-write

■ Същност при fork:

- при **клонирание на задание** първоначално Линукс ядрото забранява право на запис върху страниците на процеса родител;
- в присвоените странични таблици родителят и процесът потомък споделят едни и същи физически кадри;
- когато някой от двата процеса инициира операция по запис, се тригерира **странично прекъсване за непозволен достъп (page fault)**;
- в този случай ядрото процедира с направата на частно копие на виртуалната страница, но с вече позволен режим на запис;



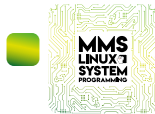
Работа с виртуална памет: обобщение

■ Предимства:

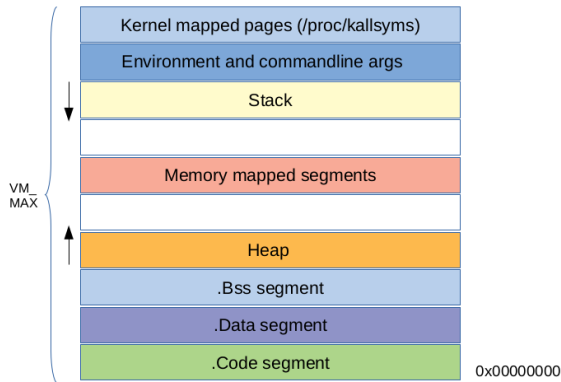
- **адресните пространства на процесите са изолирани**, като достъпът е контролиран;
- множество програми могат да има странични таблици, рефериращи **споделен регион физически кадри** (пример: динамични библиотеки);
- процеси, желаещи да организират споделен достъп до регион в паметта, биха могли да използват специализирани системни извиквания като **mmap** и **shmget**;
- колекцията системни инструменти (компилатор и свързващ редактор) не трябва да се грижат за физическата организация на паметта и нейното абсолютно адресиране



Практически пример: примерно изследване на изобразените сегменти на една С програма



Допълнена карта на сегментите



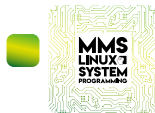
pic. based on <https://linux.die.net/man/3/etext>



Карта на сегментите: продължение

■ Обособяват се още:

- сегмент **страници, принадлежащи на ядрото**, но недостъпни за процеса, чрез които ядрото изпълнява заявени системни извиквания;
- сегмент **страници, представящи аргументи на средата и аргументи, предадени чрез команден ред**;
- края на **.text**, **.data** и **.bss** може да бъде проследен и достъпван чрез три глобални външни за програмата променливи: **etext**, **edata** и **end**



Програмен стек

■ Разновидности:

- **user**, използван за предаване на аргументи на функции, локални променливи и съхраняване на данни за контекста на всяка функция (**stack frame**) при верига на изпълнение;
- **kernel**, индивидуална област от адресното пространство на ядрото и привързана към процеса, използвана вътрешно за организация на изпълнението на системните извиквания



Функции за манипулиране на променливи на средата

■ Прототипи:

- `char *getenv(const char *name);`
- `int putenv(char *string);`
- `int setenv(const char *name, const char *value, int overwrite);`
- `int unsetenv(const char *name);`
- `int clearenv(void)`



Динамично заделяне на памет

■ Особенности:

- Линукс следва **оптимистична стратегия на заделяне на памет**, като при успешен изход от malloc, не се гарантира, че паметта ще е реално достъпна или налична;
- при недостиг на системна физическа памет се активира демонът за терминиране на процеси (**OOM killer**);
- контрол по управление на степента на натовареност на основната и вторичната памет се задават чрез: `/proc/sys/vm/overcommit_memory` и `/proc/sys/vm/oom_adj`



Динамично преместване на сегменти

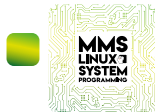
■ Налага се, когато:

- се заявяват региони споделена памет с помощта на системни извиквания като `mmap`;
- нова област динамично заделена памет се изисква при промяна на **`brk`, `sbrk`** (маркиращи края на сегмента с **неинициализирани данни**) или `malloc`;
- стекът надвиши предходно установен и позволен предишен размер



Още помощни функции

- Данни за размерите биха могли да се извлекат чрез:
 - `int` `getpagesize(void)` - архитектурно-специфична, връща стойността на **PAGE_SIZE** макродефиницията на ядрото;
 - `int` `getrlimit(int resource, struct rlimit *rlim)` - опит за прочитане на системно зададени: **размер на стек, макс. размер на даннов сегмент, макс. размер на присвоеното виртуално адресно пространство** и др.



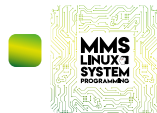
Извличане на спомагателни данни чрез procfs

■ Пример:

- `cat /proc/<pid>/status`

■ Пояснение на някои важни записи::

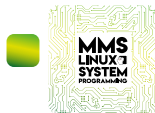
- **State**: показва текущо състояние;
- **VmPeak**: върхово достигната виртуална памет;
- **VmSize**: текущ размер на виртуалната памет;
- **VmData**: размер на данновия сегмент;
- **VmStk**: размер на потребителския програмен стеков сегмент;
- **VmExe**: размер на текстови сегмент;
- **VmLib**: размер на сегмент за изобразяване на динамични библиотечни референции;
- **VmPTE**: размер на записите в страничната таблица (PTE);
- **Threads**: брой на текущо активни нишки, подчинени на главния процес



Статистика за разпределение и ползване на паметта

```
$ cat /proc/11470/status
```

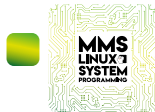
```
Name:      test
Umask:    0022
State:     S (sleeping)
...
VmPeak:    4532 kB
VmSize:    4524 kB
...
VmData:    176 kB
VmStk:     132 kB
VmExe:      8 kB
VmLib:     2120 kB
VmPTE:      56 kB
VmSwap:     0 kB
...
Threads:   1
...
```



Детайлен анализ на сегментите чрез procfs

■ Пример:

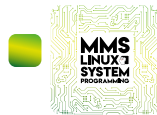
■ `cat /proc/<pid>/maps`



Пример при активен процес

```
$ cat /proc/11470/maps
```

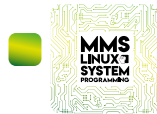
```
...
558217e9e000-558217ebf000 rw-p 00000000 00:00 0                [ heap ]
7f6e49ec3000-7f6e4a0aa000 r-xp 00000000 fd:00 7078562          / lib /
    x86_64-linux-gnu/libc-2.27.so
7f6e4a0aa000-7f6e4a2aa000 ---p 001e7000 fd:00 7078562          / lib /
    x86_64-linux-gnu/libc-2.27.so
7f6e4a2aa000-7f6e4a2ae000 r--p 001e7000 fd:00 7078562          / lib /
    x86_64-linux-gnu/libc-2.27.so
...
7f6e4a2ae000-7f6e4a2b0000 rw-p 001eb000 fd:00 7078562          / lib /
    x86_64-linux-gnu/libc-2.27.so
...
7f6e4a2b4000-7f6e4a2dd000 r-xp 00000000 fd:00 7078448          / lib /
    x86_64-linux-gnu/ld-2.27.so
...
7f6e4a4dd000-7f6e4a4de000 r--p 00029000 fd:00 7078448          / lib /
    x86_64-linux-gnu/ld-2.27.so
7f6e4a4de000-7f6e4a4df000 rw-p 0002a000 fd:00 7078448          / lib /
    x86_64-linux-gnu/ld-2.27.so
7f6e4a4df000-7f6e4a4e0000 rw-p 00000000 00:00 0
7ffe913f7000-7ffe91418000 rw-p 00000000 00:00 0                [ stack ]
```



Разяснение на извежданите данни

| Адресна граница: | Разрешения: | Офсет: | Устр. ид.: | Ид. (inode) | Абсолютен път: |
|---------------------------|-------------|--------|------------|-------------|--------------------------------------|
| 558217b8a000-558217b8c000 | r-xp | 0 | fd:02 | 14684322 | /home/users/acholakov/example_1/test |

pic. based on <https://man7.org/linux/man-pages/man5/proc.5.html>



Флагове, обозначаващи позволени операции

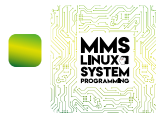
■ Описание:

- **r** - разрешено е четене;
- **w** - разрешен е запис;
- **x** - позволено е изпълнение;
- **p** - ограничен за собствен достъп (несподелен) сегмент;
- **s** - сегмент, обявен за споделен



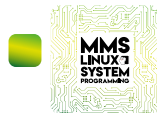
Бележки по материалите и изложението

- материалът е изготвен с образователна цел;
- съставителите не носят отговорност относно употребата и евентуални последствия;
- съставителите се стремят да използват публично достъпни източници на информация и разчитат на достоверността и статута на прилаганите или реферирани материали;
- текстът може да съдържа наименования на корпорации, продукти и/или графични изображения (изобразяващи продукти), които може да са търговска марка или предмет на авторско право - ексклузивна собственост на съотнесените лица;
- референциите могат да бъдат обект на други лицензи и лицензни ограничения;
- съставителите не претендират за пълнота, определено ниво на качество и конкретна пригодност на изложението;
- съставителите не носят отговорност и за допуснати фактологически или други неточности;
- свободни сте да създавате и разпространявате копия съгласно посочения лиценз;



Референции към полезни източници на информация

- <https://en.wikipedia.org/>
- <https://search.creativecommons.org/>
- https://en.wikipedia.org/wiki/Instruction_cycle
- https://en.wikipedia.org/wiki/Virtual_memory
- https://en.wikipedia.org/wiki/Page_table
- https://en.wikipedia.org/wiki/Memory_management_unit
- <https://pages.cs.wisc.edu/~remzi/OSTEP>
- <https://man7.org/linux/man-pages/man2/sbrk.2.html>
- <https://www.linux.com/training-tutorials/fiddling-linux-processes-memory/>
- https://refspecs.linuxfoundation.org/ELF/zSeries/lzsabi0_zSeries/x2251.html
- https://en.wikipedia.org/wiki/Kernel_page-table_isolation
- <https://software.intel.com/content/dam/develop/external/us/en/documents-tps/325462-sdm-vol-1-2abcd-3abcd.pdf>



Благодаря Ви за вниманието!

