

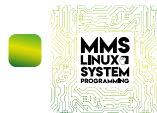
Системно програмиране за Линукс

Системни извиквания. Етапи при създаване на изпълнима програма.

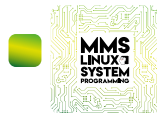
Ангел Чолаков



07.04.2021г.



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



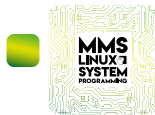
Съдържание I

- 1 Въведение
- 2 Какво е ISA, API и ABI
- 3 Режими на работа на микропроцесора
- 4 Интерфейс на системните извиквания
- 5 Етапи по създаване на изпълнима програма
- 6 Дефиниция на процес
- 7 Представяне на процеса в паметта
- 8 ELF формат
- 9 Заключение



Цел на презентацията

- Да опита да поясни:
 - какво се крие зад абривиатурите ISA, API, ABI;
 - как приложенията се възползват от услугите на ОС;
 - какво представляват системните извиквания;
 - какво е процес и как се дефинира;
 - какви са етапите при създаване на изпълнима С програма;
 - как се представя изпълнимата програма в паметта;



Въведение

■ Основни функции на ОС - предоставяне на:

- средства за заделяне и управление на системни ресурси;
- среда за изпълнение на потребителски задания (процеси);
- инструменти за подsigураване комуникация - както между изпълняваните процеси, така и между свързани в мрежа компютърни системи;
- механизми за интерактивно взаимодействие с потребителите (текстови команден интерпретатор или чрез графичен интерфейс)
- механизми за защита интегритета на потребителските данни;



Какво е ISA?

■ Спецификация, която:

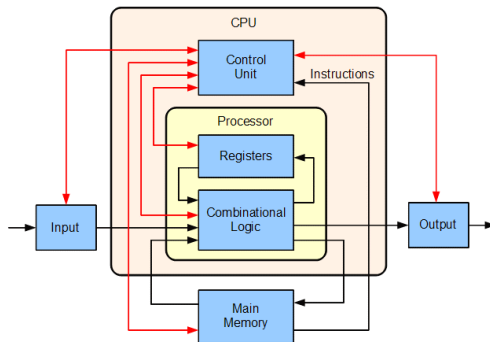
- представя набора от инструкции и абстрактния модел на изчисление за дадена компютърна архитектура;
- имплементацията на всяка такава спецификация е конкретен вид микропроцесор (CPU);

■ Наборът от инструкции показва:

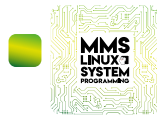
- какви процесорни регистри са налични;
- какви са машинните команди и типовете данни;
- как се изгражда моделът на паметта и какви са режимите за адресация;
- как е организирана обработката на прекъсванията и др.



Опростена блокова структура на микропроцесор



pic. by Lambtron , CC BY-SA 4.0 via Wikimedia Commons



Какво е API?

■ Интерфейс, чрез който:

- разработчикът се възползва от предоставен набор функции в процеса на създаване на приложение с помощта на език от високо ниво;
- реализацията на тези функции остава скрита, като тя може да е поверена на съставни блокове от самата ОС или да е пакетизирана под формата на системни библиотеки посредници;

API включва само описание на прототипа на наличните функции заедно с броя и формата на формалните параметри и типа на връщания резултат.



Какво е ABI?

■ Конвенция, която показва:

- как структурите от данни и/или машинните подпрограми в контекста на дадена процесорна микроархитектура се достъпват на ниско ниво;

■ Този интерфейс дефинира още и:

- как са организирани самите изчисления;
- как се форматират входните и изходните данни (предавани аргументи);
- как става представянето на данните в регистрите на процесора или паметта;
- какъв е редът на разполагане на данните и др.

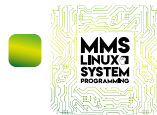


Основни режими на работа на микропроцесора

■ Привилегирован (supervisor) Разрешени са:

- изпълнението на пълния набор от процесорни инструкции;
- достъпът до системните шини и периферни устройства, като е възможен и невъзпрепятстван достъп до цялата наличната физическа памет;
- прецизно управление на обслужването на хардуерните прекъсвания и/или тяхното динамично забраняване или разрешаване

Това е основният режим, в който ОС работи с процесора.



Основни режими на работа на микропроцесора

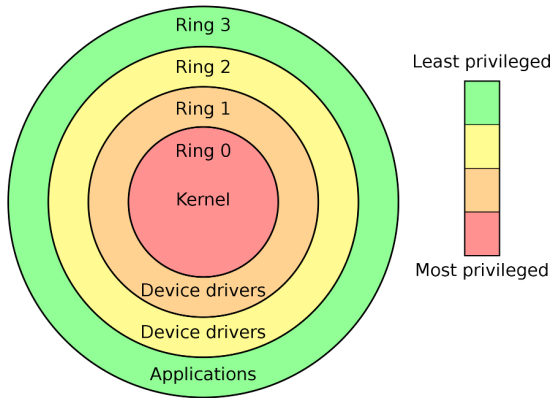
■ Потребителски (user)

- прилагат се принципите на изолация на изпълняваните задачи чрез поставянето им в отделени един от друг и защитени региони от адресното пространство на паметта;
- избягва се непосредствена работа с физически адреси;
- въвежда се логическо обособяване на виртуални региони от паметта, които са под контрола на обособен транслиращ хардуер - мениджър на виртуална памет (MMU);
- директният достъп до паметта или до определено системно устройство са забранени;
- не всички управляващи инструкции и регистрови структури на процесора са активни

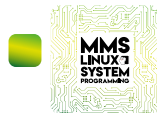
В този режим се изпълняват потребителските процеси или задания.



Пример за разслюяване на поддържаните режими



pic. by Hertzsprung, CC BY-SA 3.0 via Wikimedia Commons



Интерфейс на системните извиквания

■ Същност:

- **набор от подпрограми на една ОС**, чрез които тя предоставя своите услуги на потребителските задания съобразно конкретиките на всяка микроархитектура;
- съществува силна корелация между множеството от функции, което изгражда един приложен програмен интерфейс (API) и набора от асоциираните системни функции или извиквания;
- **корелацията** е представена посредством пакет от системни библиотеки, които се предоставят като част от инструментите за разработка на даден програмен език, включващи: **компилатор, свързващ редактор и зареждаща програма**



Механика на системните извиквания

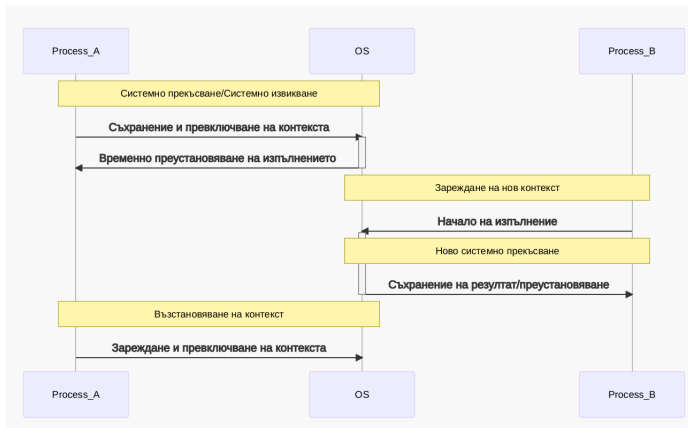
- **Изпълнението** на системните извиквания обикновено се съпътства:
 - с **превключване** на режима на работа на процесора;
 - изпълнение на поредица подпрограми на ОС в привилегирован режим от името на извикващия процес;
 - връщане на резултат от обработката чрез повторно превключване в потребителски режим на работа

Не е задължително това превключване да стане веднага с извикването на помощната библиотечна функция - това се случва по платформено зависим начин след предаване контрол върху процесите на операционната система.

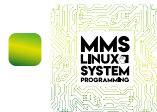
Прибягва се до т.нар. механизъм на **софтуерни прекъсвания** или специализирани инструкции, които тригерират събитие по софтуерно прекъсване на работата на процесора и **превключване на контекста**.



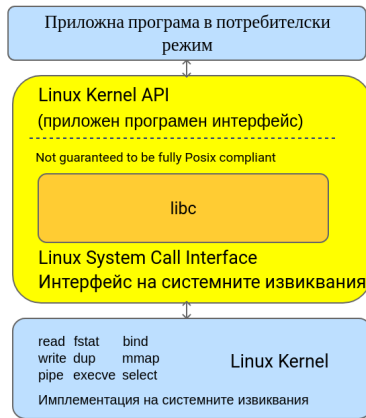
Процедура по системно извикване - илюстрация



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



Диаграма на системните извиквания в Линукс



pic. based on work by Shmuel Csaba Otto Traian, CC BY-SA 4.0 via Wikimedia Commons



Категории системни извиквания

- Групи в контекста на Линукс ядрото:
 - създаване и управление на потребителски процеси: **fork**, **execv**, **kill**, **exit** и др.;
 - работа с файлове, файлови атрибути и файлови дескриптори: **open**, **read**, **write**, **lseek**, **close** и др.;
 - заявяване и освобождаване на системни устройства и управление на устройствени атрибути: **read**, **write**, **ioctl**;
 - поддържащи функции и средства за контрол върху поведението на системата: **sched_get_affinity**, **sched_setaffinity**, **getpid** и др.;
 - комуникация между процеси и системи: **pipe**, **mmap**, **socket** и др.;
 - механизми за защита на достъпа и задаване на правомощия: **chmod**, **chown**, **umask** и др.



Практически пример: проследяване на системните ИЗВИКВАНИЯ

```
strace -o sc_trace.txt -f -x -c ./test_util
```

% time	seconds	usecs / call	calls	errors	syscall
0.00	0.000000	0	3		read
0.00	0.000000	0	2		write
0.00	0.000000	0	1		fstat
0.00	0.000000	0	4		brk
0.00	0.000000	0	1	1	access
0.00	0.000000	0	1		execve
0.00	0.000000	0	1		uname
0.00	0.000000	0	1		readlink
0.00	0.000000	0	1		arch_prctl
0.00	0.000000	0	1		openat
100.00	0.000000		16	1	total



Процедура по създаване на изпълнима С програма

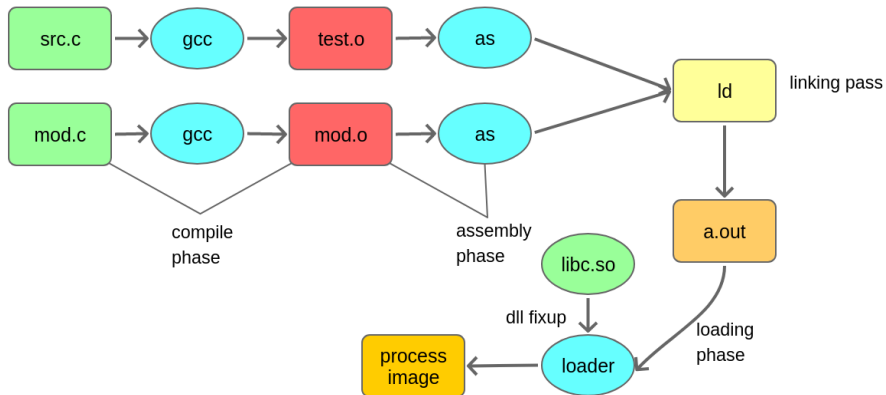
■ Фази, илюстриращи процеса:

- 1 **създаване** и редактиране на изходния код на програмата;
- 2 **компилиране**, при което текстовото описание се преобразува до последователност от машинни инструкции съгласно конвенцията на приложния бинарен интерфейс (ABI);
- 3 **асемблиране и свързване**, което има за цел да разчете символните таблици и разреши правилно крос-реферирането на даннови структури в колекцията от обединявани обектни файлове. Свързването бива статично или динамично.

Резултатът довежда до създаване на машинно съвместим изпълним файл. Той обаче е пасивен обект, представлящ списък от инструкции с определено предназначение, съхранени на външен носител.



Диаграма, илюстрираща процедурата

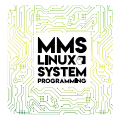


pic. based on <https://en.wikipedia.org/wiki/Compiler>



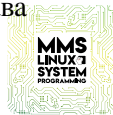
Защо няма да коментираме тези фази детайлно сега?

- **Парсване и Компилиране**, защото:
 - изискват специализирани познания от дисциплини като **дискретна математика, информатика, създаване на програмни среди и технология на програмирането**;
- **Асемблиране и свързване**, защото:
 - предполагат познания за това как се разпределя и управлява **адресното пространство** на паметта и какви механизми за работа с **виртуална памет** съществуват;

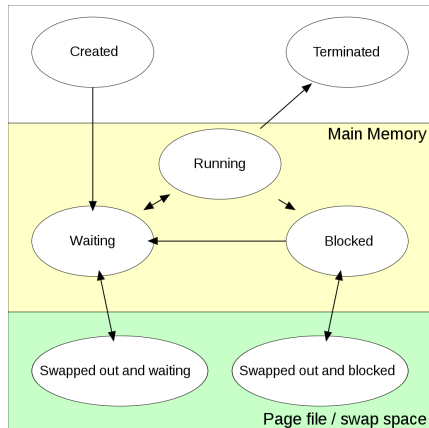


Дефиниция на процес

- Представява **програма в етап на изпълнение**. Дефиницията включва:
 - съвкупността от региони виртуална памет, в които се изобразява съдържанието на изпълнимия файл;
 - атрибути и ресурси, които ОС назначава, за да диспечерира изпълнението на програмата във времето;
 - наборът от използваните процесорни регистри, тяхното състояние и съдържание във времето, а също и ползвания режим на адресиране на физическата памет;
 - йерархия от производни подпроцеси или подзадачи, ако процесът създава или е създал такива



Състояния на един процес



pic. by A3r0 assumed, Public Domain via Wikimedia Commons



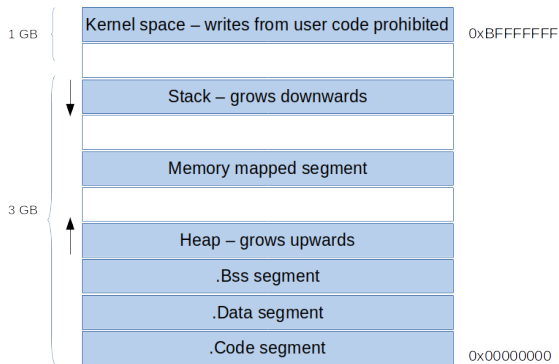
Състояния на процес

■ Обособяват се типично:

- **новосъздаден** - след зареждане на изпълнимия файл в паметта и преди изпълнение на машинни инструкции;
- **чакащ изпълнение** - в очакване на разрешение за изпълнение от страна на диспечера;
- **изпълняващ се** - в етап на изпълнение на последователността от машинни инструкции;
- **блокиран** - при заявка на достъп до физическо устройство или обмен на данни с друг процес. Управлението се предава на друга програма и текущият процес преминава в чакащо състояние докато не бъде уведомен от диспечера;
- **терминиран** - при завършване на изпълнението или терминиране от страна на ОС;



Опростено представяне на процеса в паметта



pic. based on work by Surueña, CC BY-SA 3.0 via Wikimedia Commons



Представяне на процес: вътрешно описание

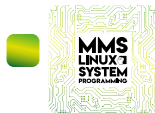
■ Най-важни сегменти:

- **програмен стек (stack)**;
- сегмент за рефериране на споделени региони от паметта **mmap**;
- раздел за **динамично заделяна памет (heap)**;
- **.bss сегмент** - тук се разполагат неинициализираните глобални и статични променливи;
- **.data** или даннов сегмент - тук се пазят инициализираните глобални и статични променливи;
- **.text** или текстови сегмент, който отразява последователността от машинни инструкции, които описват логиката на програмата;



Предназначение на програмния стек

- Реализира операции, свързани с:
 - предаване и прочитане на аргументи при извикване на функция;
 - отразяване състоянието и обработката на локални променливи и прочитане на изходния резултат от изпълнението на функция;
 - проследяване на последователността от извиквани функции и техния контекст в динамиката на изпълнение на програмата



ELF file format: въведение

Описва пакетирането на програмни задания в изпълним файлов контейнер и изграждане на изпълним образ на процеса в паметта.

- Включва в състава си:

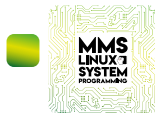
- **сегменти** - раздели, които указват области от изпълнимия файл, които съдържат необходими за изпълнението на програмата елементи и участват непосредствено в изграждането на изпълним образ на процеса в паметта;
- **секции** - подраздели, съставна част от отделните сегменти. Секциите се прочитат и обработват преди и по време на фазата на свързване (linking)



ELF file header

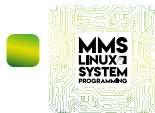
■ Предназначение:

- съдържа характеристики метаданни, чрез които идентифицира изпълнимия файл и указва как той се прочита и интерпретира от конкретна машинна архитектура



ELF file header

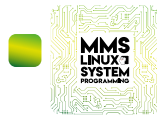
- Някои от най-съществените характеристики са:
 - тип на файла;
 - кодиране на данните в обектния формат;
 - разновидността микропроцесорна архитектура;
 - адреса в паметта на стартовата точка на изпълнение;
 - начало на таблицата със заглавни програмни раздели (program header table);
 - начало на таблицата със заглавни секции на програмата



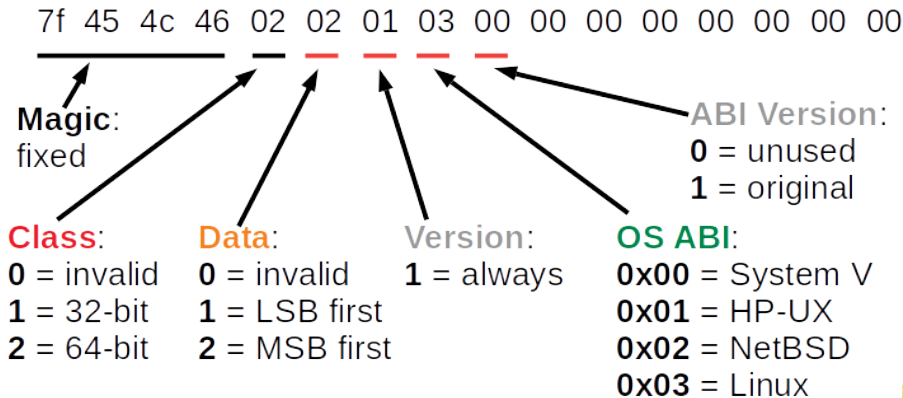
ELF file header: формат

```
//C
```

```
typedef struct {
    Byte          e_ident[16];
    HalfWord      e_type;           /* Common types          */
    HalfWord      e_machine;       /* 0  = ET_NONE          */
    Word          e_version;       /* 1  = ET_REL           */
    Address       e_entry;         /* 2  = ET_EXEC          */
    Offset        e_phoff;        /* 3  = ET_DYN           */
    Offset        e_shoff;        /* 4  = ET_CORE          */
    Word          e_flags;         /* .....                */
    HalfWord      e_ehsize;
    HalfWord      e_phentsize;
    HalfWord      e_phnum;
    HalfWord      e_shentsize;
    HalfWord      e_shnum;
    HalfWord      e_shstrndx;
} ELFHeader
```



ELF file header: идентифициране



pic. based on <https://refspecs.linuxfoundation.org/elf/elf.pdf>

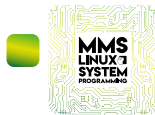




Заглавен програмен раздел (program header)

Служи за дефиниране на всеки един от съставните програмни сегменти. Таблицата със заглавни програмни раздели се намира, отместена на **e_phoff** спрямо началото на файла и се състои от **e_phnum** записа, всеки с размер **e_phentsize**.

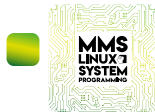
Размерът на съставните полета за всеки запис се различава в зависимост от това дали форматът на ELF файла таргетира 32 или 64-битова архитектура.



Заглавен програмен раздел (program header)

■ Някои от най-важните полета са:

- тип на представяния сегмент;
- отместване на сегмента в изпълнимия файлов образ;
- виртуален адрес на сегмента;
- физически адрес в случаите, когато това е от значение;
- флагове за достъп - маркиран за четене (PF_R), запис (PF_W) или съдържащ изпълними инструкции (PF_X);



ELF program header: формат

```
//C
```

```
typedef struct {  
    Word    p_type;           /* Common types          */  
    Offset   p_offset;  
    Address  p_vaddr;         /* 0 = PT_NULL;          */  
    Address  p_paddr;         /* 1 = PT_LOAD;           */  
    Word     p_filesz;        /* 2 = PT_DYNAMIC;        */  
    Word     p_memsz;         /* 3 = PT_INTERP;         */  
    Word     p_flags;         /* 4 = PT_NONE;           */  
    Word     p_align;         /* 5 = PT_SHLIB;          */  
} ProgramHeader32           /* 6 = PT_HEADER;         */  
                             /* 7 = PT_TLS;            */  
                             /* ...                    */
```



ELF program header: типове сегменти

■ Пояснение:

- PT_NULL - неактивен сегмент;
- PT_LOAD - зареждаем в паметта сегмент;
- PT_DYNAMIC - указва как да се организира зареждането на динамични библиотеки;
- PT_INTERP - пази данни за абсолютен път, идентифициращ програмен интерпретатор;
- PT_NOTE - съдържа допълнителни спомагателни данни;
- PT_SHLIB - резервиран, но с неспецифицирано значение;
- PT_PHDR - специфицира локацията и размера на таблицата със заглавни сегменти;
- PT_TLS - показва, че сегментът съдържа TLS (thread local storage) данни. Това са статични или глобални променливи в контекста на дадена нишка, а не в контекста на всички системни потоци от задачи. Пример: **errno**;



Практически пример: прочитане на програмните сегменти

```
readelf --segments ~/Desktop/Test/Project/test_util
```

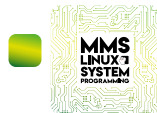
Elf file `type` is EXEC (Executable file)

Entry point 0x400a50

There are 6 program headers, starting at offset 64

Program Headers:

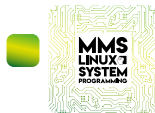
Type	Offset FileSiz	VirtAddr MemSiz	PhysAddr Flags Align
LOAD	0x0000000000000000 0x00000000000b5e0e	0x0000000000400000 0x0000000000b5e0e	0x0000000000400000 R E 0x200000
LOAD	0x00000000000b6120 0x0000000000051b8	0x000000000006b120 0x000000000006920	0x000000000006b120 RW 0x200000
NOTE	0x0000000000000190 0x0000000000000044	0x00000000000400190 0x0000000000000044	0x00000000000400190 R 0x4
TLS	0x00000000000b6120 0x0000000000000020	0x000000000006b120 0x0000000000000060	0x000000000006b120 R 0x8
GNU_STACK	0x0000000000000000 0x0000000000000000	0x0000000000000000 0x0000000000000000	0x0000000000000000 RW 0x10
GNU_RELRO	0x00000000000b6120 0x0000000000002ee0	0x000000000006b120 0x0000000000002ee0	0x000000000006b120 R 0x1



Заглавна част на секция (section header)

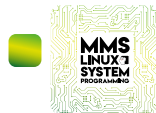
Служи за дефиниране на всяка една от съставните секции.

Разделите са подредени в масив, наречен таблица със заглавни части на секциите.



Заглавна част на секция (section header)

- Някои от най-важните полета са:
 - име на секцията - офсет в таблицата със символни низове;
 - тип на секцията;
 - флагове за достъп;
 - адрес, на който секцията се изобразява в образа на процеса в паметта, ако съдържанието е част от зареждаем сегмент;
 - офсет - локацията на данните в ELF файла;
 - индекс на асоциирана секция;
 - допълнителна информация;



ELF section header: формат

```
//C
```

```
typedef struct {
    Word    sh_name;           /* Common types          */
    Word    sh_type;           /* 0  = SHT_NULL          */
    Word    sh_flags;          /* 1  = SHT_PROGBITS      */
    Address sh_addr;           /* 2  = SHT_SYMTAB        */
    Offset  sh_offset;         /* 3  = SHT_STRTAB        */
    Word    sh_size;           /* 4  = SHT_RELA          */
    Word    sh_link;           /* 5  = SHT_HASH          */
    Word    sh_info;           /* 6  = SHT_DYNAMIC       */
    Word    sh_addralign;      /* 7  = SHT_NOTE          */
    Word    sh_entsize;        /* 8  = SHT_NOBITS        */
} SectionHeader32             /* 9  = SHT_REL           */
                               /* 10 = SHT_SHLIB         */
                               /* 11 = SHT_DYNSYM        */
                               /* .....                */
```



ELF section header: типове секции

■ Пояснение:

- SHT_NULL - неидентифицирана секция;
- SHT_PROGBITS - данни или код за програмата;
- SHT_SYMTAB - таблица със структури, дефинираща използвани символи;
- SHT_STRTAB - таблица със символни низове;
- SHT_RELA - показва как се модифицират различните секции при асемблиране;
- SHT_HASH - хеш таблица на символите;
- SHT_DYNAMIC - данни при динамично свързване и рефериране на библиотечни функции;
- SHT_NOTE - спомагателни данни от разнороден характер;
- SHT_NOBITS - секция с неинициализирани променливи;
- SHT_REL - показва как се модифицират различните секции при асемблиране;
- SHT_SHLIB - резервирана, но с неспецифирано приложение;
- SHT_DYNSYM - таблица със структури, обозначаваща динамично реферирани символи, обработвани от свързващия редактор;



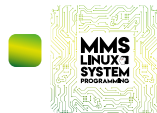
Практически пример: прочитане на програмните секции

```
readelf --sections ./test_util
```

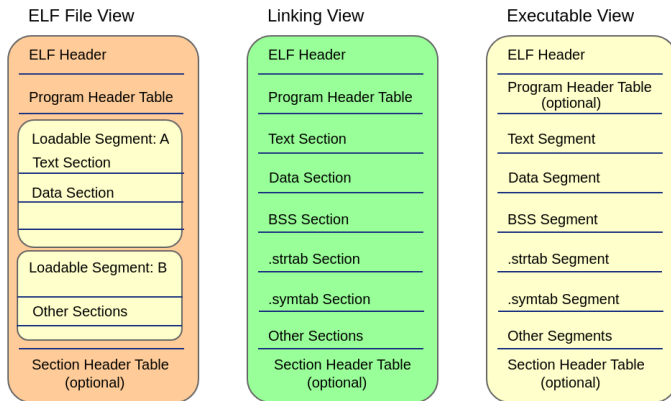
```
...
```

```
Section Headers:
```

[Nr]	Name Size	Type EntSize	Address Flags Link Info	Offset Align
	0000000000000228	0000000000000018	AI 0 20	8
[4]	.init	PROGBITS	0000000000400400	00000400
	0000000000000017	0000000000000000	AX 0 0	4
...				
[10]	.rodata	PROGBITS	0000000000492440	00092440
	000000000000192ac	0000000000000000	A 0 0	32
...				
[21]	.data	PROGBITS	00000000006b90e0	000b90e0
	0000000000001af0	0000000000000000	WA 0 0	32
...				
[26]	.bss	NOBITS	00000000006bb2e0	000bb2d8
	0000000000001738	0000000000000000	WA 0 0	32
...				
[30]	.symtab	SYMTAB	0000000000000000	000bc940
	000000000000aae8	0000000000000018	31 680	8



ELF file: обобщение - третиране на сегменти и секции



pic. based on https://www.keil.com/support/man/docs/armlink/armlink_pge1406297322750.htm



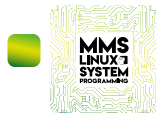
Процедура по зареждане в Линукс - фаза I

- Опростена последователност:
 - **командният интерпретатор** прочита ELF файла и се пристъпва към системно извикване (**execve**), чрез което контролът се подава на ядрото;
 - **заглавните байтове на ELF файла** определят кой вграден в ядрото обработчик (**handler**) да се извика;
 - (**fs/binfmt_elf.c -> load_binary**) се изпълнява, за да разчете ELF файла



Процедура по зареждане в Линукс - фаза II

- Опростена последователност:
 - обработчикът в ядрото се грижи за **първоначалната конфигурация на паметта**, в която сегментите ще бъдат заредени;
 - ако се намери **.interp** секция, ядрото изобразява външната зареждаща програма в паметта и я стартира, за да може тя да продължи с обработката на изпълнимия файл;
 - ако не се намери .interp секция, ядрото зарежда ELF файла в паметта непосредствено и го изпълнява



ELF file: обобщение

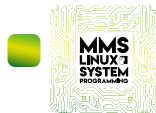
■ Особенности:

- не е задължително размерът на един сегмент в ELF файла да кореспондира на размера на областта от паметта, в която този сегмент се поставя и изобразява;
- стартовият адрес на програмата обикновено обозначава стартовия адрес на **програмния интерпретатор**, който обезпечава среда за действителното изпълнение на програмата;
- има още много подробности, които са описани в реферираната документация;
- **динамичното свързване и механизмите на свързване** заслужават отделна презентация



Какво предстои да разглеждаме по-натам?

- Политики за управление на процеси;
- Виртуална памет и механизми на изобразяване на сегментите на един изпълним файл;
- Очертаем механизмите на свързване и зареждане



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/Process_state
- https://en.wikipedia.org/wiki/Linux_kernel_interfaces
- <https://man7.org/linux/man-pages/man2/syscalls.2.html>
- https://web.archive.org/web/20100218115342/http://www.linfo.org/context_switch.html
- https://en.wikipedia.org/wiki/Computer_architecture
- https://en.wikipedia.org/wiki/Instruction_set_architecture
- https://en.wikipedia.org/wiki/Executable_and_Linkable_Format
- <https://refspecs.linuxfoundation.org/elf/elf.pdf>
- <https://elixir.bootlin.com/linux/latest/source/include/uapi/linux/elf.h>
- <https://lwn.net/Articles/631631/>
- https://www.keil.com/support/man/docs/armlink/armlink_pge1406297322750.htm



Благодаря Ви за вниманието!

