

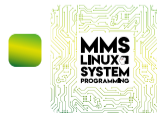
Системно програмиране за Линукс

Операционни системи: предназначение и организация

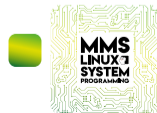
Ангел Чолаков



19.03.2021г.

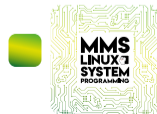


This work is licensed under a Creative Commons
“Attribution-ShareAlike 4.0 International” license.



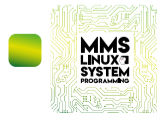
Съдържание I

- 1 Въведение
- 2 Исторически етапи
- 3 Дефиниция на ОС
- 4 ОС и вградените системи
- 5 Цели при разработката на ОС
- 6 Разновидности ОС
- 7 Структура на Линукс ядрото
- 8 Практическа секция - GNU Make
- 9 Заключение



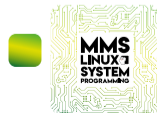
ВЪВЕДЕНИЕ

- Какво е операционна система?
 - неделима част от програмното осигуряване на една компютърна система;
 - грижи се за управлението на системните ресурси и обезпечаването на среда за потребителско взаимодействие.



Цел на презентацията

- Да опита да поясни:
 - как се дефинира понятието ОС;
 - какви са основните цели в процеса на разработка на ОС;
 - какво е различието между политика и механизъм;
 - посочи разновидности ОС според заложените политики и механизми;
 - очертае структурната организация на Линукс ядрото;
 - постави основите на автоматизиране на процеси по създаване на приложения с GNU Make

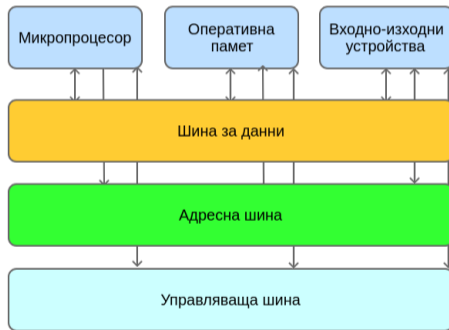


Исторически етапи в разработката на ОС

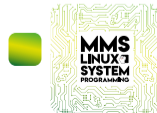
- Предназначение на **монитор** програмата
 - зареди в паметта програмни задания;
 - стартира изпълнението им и изведе резултат;
- Еволюционно развитие
 - мониторът се преработва из основи и се разраства съобразно символните машинни езици;
 - вграждат се редица функции по обработка на системни прекъсвания, поддържане на многозадачен режим на работа и управление на паметта;
 - програмното обезпечаване се допълва чрез колекция системни приложения: **компилатор, свързващ редактор и интерактивен интерпретатор** на команди



Илюстрация на Фон-Ноймановата архитектура

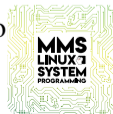


pic. based on work by W. Nowicki, CC BY-SA 3.0 via Wikimedia Commons



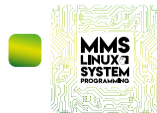
Дефиниция на ОС

- От гледната точка на **системните архитекти**:
 - подsigурява управлението на привързаните хардуерни компоненти посредством драйвери (подсистеми в състава на ОС, които се третират като форма на комуникационен и управляващ интерфейс);
 - менажира и координира достъпа до наличните ресурси, като предоставя механизми за арбитраж и решаване на конфликти;
 - подsigурява среда за последователно или паралелно изпълнение на множество потребителски задачи, наречени процеси;
 - гарантира изолация между програмите и данните на различните потребители - както по отношение на адресното пространство, така и по отношение на правомощията за достъп



Дефиниция на ОС

- От гледната точка на **потребителите**:
 - ядро и колекция от свързани системни програми;
 - ядрото се възприема като контролна програма, чието изпълнение е непрекъснато и жизненоважно за стартирането и управлението на системни и потребителски задачи;
 - изпълняваните приложни програми се възползват от функциите на операционната система посредством програмни интерфейси и системни библиотеки-посредници;



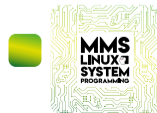
Режими на работа със системните ресурси

■ Привилегирован:

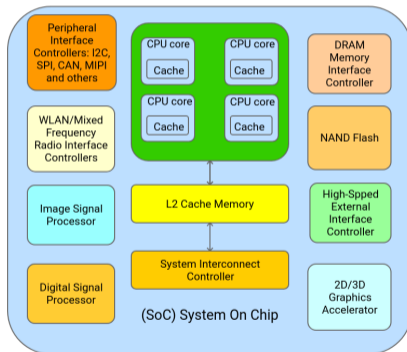
- kernel mode - режим на ядрото, в който се позволява работа с пълния набор ресурси и достъпът до физическата памет е без ограничения;

■ Потребителски:

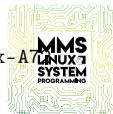
- user mode, при който всеки процес се изпълнява в заделена и изолирана област от паметта без директен достъп до функционалността на хардуера;



ОС в контекста на една вградена система днес



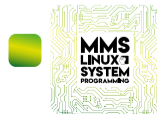
pic. based on publicly available ARM-based SoC descriptions, e.g.: https://en.wikipedia.org/wiki/ARM_Cortex-A7



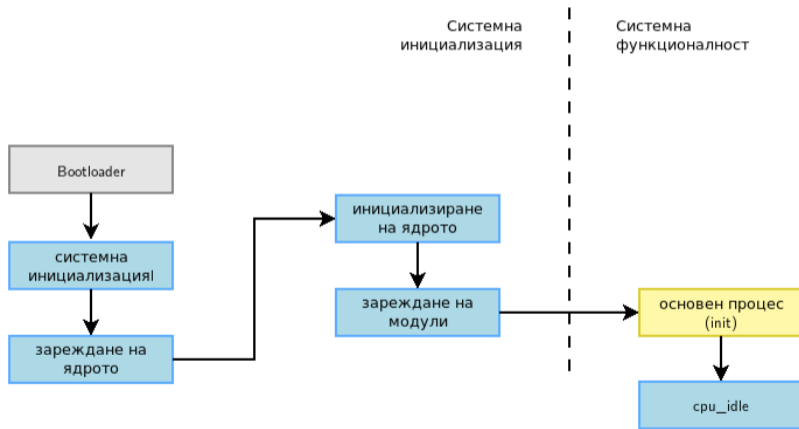
Структура на една система върху чип (SoC)

■ Основни блокове:

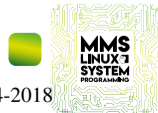
- микропроцесорен блок (обикновено RISC-базиран) с няколко процесорни ядра;
- блок, който подsigурява бърз вторичен кеш;
- контролер за управление на паметта (MMU + DRAM controller);
- бърза системна шина с поддръжка за динамично управление на захранването и тактовите честоти на отделните функционални звена;
- графичен процесор за ускоряване на работата с приложения, свързани с тримерна графика (GPU);
- опционален копроцесор за обработка на двумерни изображения и камера сензори (ISP);
- опционален копроцесор за обработка на цифрови сигнали - за целите на аудио/видео обработки, кодеци и др. (DSP);
- периферни контролери за менажиране на разнообразни входно-изходни физически интерфейси като I2C, SPI, USB и др.;
- малък блок EPROM/FLASH памет за съхранение на конфигурационни данни;
- други специализирани блокове за обезпечаване на телекомуникационни функции;



Последователност по стартиране (bootstrapping)

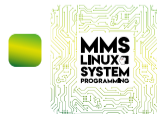


pic. based on <https://bootlin.com/doc/legacy/kernel-init/src/> by Bootlin, CC BY-SA 3.0, 2004-2018



Цели при разработката на ОС

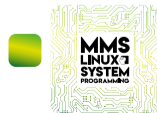
- От страна на **системните разработчици**:
 - модулна структура и функционално разделяне на отделни специализирани блокове;
 - ясно дефиниране на интерфейса за всеки един от тези блокове;
 - осигуряване на механизми за сигурен обмен на данни;
 - възможности за конфигуриране на системата и гъвкаво надграждане с нови модули;
 - наличие на механизми за мониторинг и отстраняване на дефекти;



Цели при разработката на ОС

■ От страна на потребителите:

- постигане на оптимално оползотворяване на машинните ресурси и максимална производителност;
- лекота при употреба в процеса на интерактивно взаимодействие;
- обезпечаване на надеждна и сигурна работа с минимум проявяващи се дефекти;



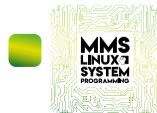
Политика и механизъм

■ Политика

- указва как се изменя и адаптира поведението на съставните блокове;

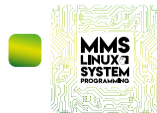
■ Механизъм

- описва как се реализира дадената цел или политика (визира се програмното описание)



Разновидности ОС: Компактни

- Характеристики:
 - характеризират се с резидентен управляващ процес;
 - притежават тясно специализирана логика;
 - разполагат с фиксиран набор механизми на контрол;
 - често пъти липсва и възможност за многозадачна и многопотребителска работа



Компактни ОС: организация



pic. inspired by <http://en.wikipedia.org/wiki/Image:OS-structure.svg> by Golftheman



Разновидности ОС: Монолитни

■ Характеристики:

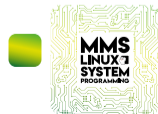
- работят в привилегирован режим на процесора и поемат подsigуряването на всички системни функции;
- достъпът от страна на приложенията до така предоставяните услуги става чрез подробно описан системен интерфейс и библиотеки-посредници. Чрез последните приложенията могат да отправят заявките си под формата на системни извиквания, които ОС обработва;
- приложенията се развиват в отделна област от паметта и не споделят непосредствено адресно пространство с процесите на операционната система;



Монолитни ОС: организация



pic. inspired by <http://en.wikipedia.org/wiki/Image:OS-structure.svg> by Golftheman



Разновидности ОС: С микроядро

■ Характеристики:

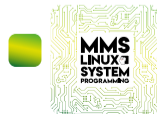
- на микроядрото е поверено само обслужването на най-важните системни функции като управление на паметта, менажиране на процеси и междупроцесна комуникация;
- останалите подсистеми на ОС, включително драйверите за обслужване на периферията, са изнесени извън ядрото под формата на отделни потребителски процеси, които общуват, като разменят съобщения помежду си;
- отработването на потребителските заявки става посредством непрекъснатата размяна на съобщения и обикновено това може да се окаже тясно място в оценката на производителността;



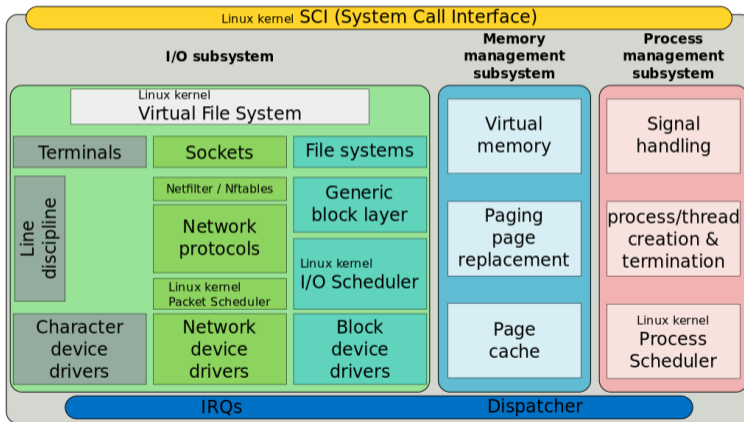
ОС с микроядро: организация



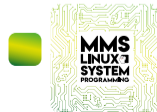
pic. inspired by <http://en.wikipedia.org/wiki/Image:OS-structure.svg> by Golftheman



Структура на Линукс ядрото



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



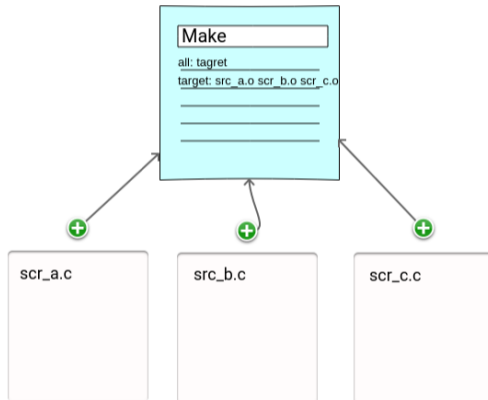
Структура на Линукс ядрото

■ Основни подсистеми:

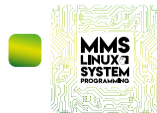
- **интерфейсът на системните извиквания (SCI)** - играе ролята на посредник в комуникацията с приложенията. Той спомага за предоставяне на достъп на приложенията до услугите на ОС;
- **диспечерът на процеси** - той е отговорен за изпълнението на системните задачи и процесите, представящи потребителски задания;
- **подсистема за управление на паметта** - често пъти обвързана с концепцията за виртуална памет и нейното динамично управление;
- **мрежова подсистема** - обезпечават функции по транспорт и обмяна на съобщения посредством мрежови комуникационни интерфейси;
- **файлова подсистема** - грижи се както за детекция и управление на блоково-ориентирани входно-изходни устройства, така и за форматирането на заявките за работа с файлови структури;
- **архитектурна и платформена подсистема** - подsigурява машинна поддръжка за дадена процесорна архитектура;
- **колекция периферни и платформени драйвери** - грижат се за управлението на хардуерната част на ниско ниво и взаимодействието с различни физически устройства



Практическа секция: Първи стъпки с GNU Make

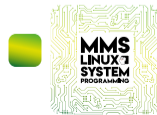


pic. based on public GNU Make documentation



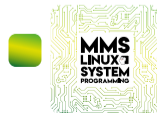
Как е организирано изграждането на софтуерни проекти?

- Детайлно описание на:
 - списъка от файлове, които представляват изходния код на един софтуерен проект;
 - групирането на тези файлове в поддиректории и логическата йерархия помежду им;
 - условни зависимости и параметри, повлияващи изходния резултат;
 - указване на инструментите, които участват в процедурата по изграждане на проекта и съпътстващите управляващи настройки на средата



Характеристики на GNU Make

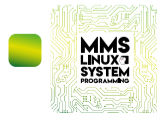
- **С приложение**, което:
 - прочита, обработва и изпълнява набор от команди, разкриващи структурата на един проект и стъпките по генерирането на изпълним код;
 - е способно да проследява изменения в изходния код;
 - изпълнява само тези последователности от операции, които имат връзка с направените изменения;
 - разчита на декларативен макро подобен скриптов език за описание на управляващите обработки



GNU Make: синтаксис

■ Особености:

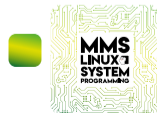
- използва се **декларативен език**, чрез който се описват поредица от задачи;
- всяка задача е представена чрез посочване на текстови идентификатор, наречен **цел (target)** и списък от **зависимости (подзадачи)**;
- операциите, които обозначават дадена задача, се изреждат под формата на поредица от текстови команди непосредствено след посочване на целта;
- съставните операции в рамките на една задача се разграничават синтактично с помощта на символ за табулация



Нашият първи Makefile

Makefile

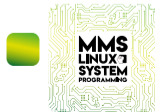
```
hello :  
    @echo "Hello Make!"
```



GNU Make: механика на парсване

■ Особенности:

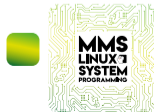
- парсването е организирано на **два етапа**;
- **първата фаза** включва обработка на изразите, присвоявани на променливи и разчитане на задачите;
- резултат от тази първа фаза е **граф на зависимостите**;
- **втората фаза** използва междинните вътрешни данни от предходната, за да определи кои задачи и съставни операции да бъдат изпълнени



GNU Make: изпълнение

■ Същност на механизма:

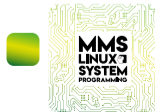
- 1 **парсване** съгласно описания ред;
- 2 **изпълнение** на задачите според графа на зависимостите;
- 3 за всяка операция на отделен ред в рамките на задача се стартира нова инстанция на командния интерпретатор (shell);
- 4 разширяване на зависимите стойности за динамично определяни променливи в процеса на изпълнение;



GNU Make: грешки по време на изпълнение

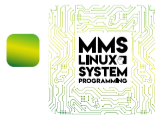
■ Подход:

- при наличие на грешка изпълнението на текущата задача се прекъсва;
- при наличие на грешка, ако не е указано друго за конкретна операция, **make преустановява изпълнението** на останалите неизпълнени задачи и връща код за грешка;
- ако резултатът от дадена съставна операция не е определящ, евентуална грешка може да се игнорира чрез ”-” в началото на ред



GNU Make: синтаксис

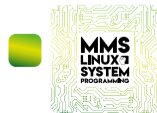
- Основни типове задачи:
 - такива, пряко свързани с наличието на сорс файлове;
 - процедурни, при които не е необходимо да се посочват зависими файлове
.PHONY;
 - задачи по подразбиране .DEFAULT_GOAL



Нека да добавим още задачи

Makefile

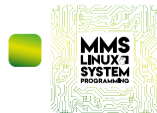
```
hello :  
    @echo "Hello Make"  
  
create :  
    @echo "Create a .c file , if it does not exist!"  
    touch test.c  
  
clean :  
    @echo "Cleaning test.c file"  
    -rm -f *.c
```



Избор на задача по подразбиране

Makefile

```
.DEFAULT_GOAL := create  
.PHONY: all hello create clean  
  
all: hello  
  
hello:  
    @echo "Hello Make"  
  
create:  
    @echo "Create a .c file , if it does not exist!"  
    touch test.c  
  
clean:  
    @echo "Cleaning test.c file"  
    rm *.c
```



Първи Makefile за компилиране на C програма

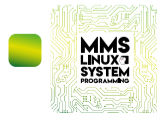
Makefile

```
.PHONY = all clean

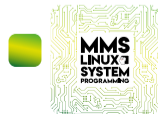
# The default target relates to the
# test executable we want to generate
all: test

# Rule to compile the test executable
test: test.c test.h
    @echo "Compiling output"
    gcc -o test -lm test.c test.h

# Rule to clean the generated files
clean:
    @echo "Cleaning the test app"
    -rm -f test test.o
```



А има ли начин да сме по-ефикасни?



GNU Make: синтаксис

■ Променливи:

- рекурсивно представяни **CC = gcc**;
- такива, при които рекурсивни референции са забранени **CC := gcc**

■ Някои специални такива:

- **CC**: указва програма за компилиране;
- **CFALGS**: указва флагове, ползвани при компилация;
- **LIBS**: обозначава свързани библиотечни референции;
- **\$@**: обозначава обекта отляво на **:** в едно правило;
- **\$<**: обозначава първия елемент от списъка със зависимости отдясно на **:** в едно правило;
- **\$\$**: обозначава всички елементи в списъка със зависимости



Примерен Makefile с променливи

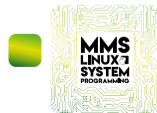
Makefile

```
.PHONY = all clean
CC = gcc
LFLAGS = -lm

test: test.o
    @echo "Compiling output"
    ${CC} ${LFLAGS} $< -o $@

test.o: test.c test.h
    @echo "Creating object files"
    ${CC} -c $<

clean:
    @echo "Cleaning build products"
    -rm -vf test.o test
```



По-сложен пример за компилиране на С програма

Makefile

```
.PHONY = all clean
```

```
CC = gcc
```

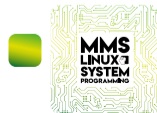
```
LFLAGS = -lm
```

```
SRC_LIST := $(wildcard *.c)
```

```
OBJ := $(SRC_LIST:%.c=%.o)
```

```
OUT := $(SRC_LIST:%.c=%)
```

```
all : ${OUT}
```



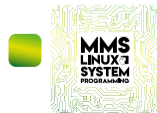
Компилиране на C програма: продължение

Makefile

```
${OUT}: ${OBJ}  
    @echo "Compile final output"  
    ${CC} ${LFLAGS} $< -o $@
```

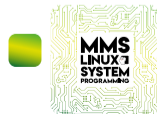
```
${OBJ}: ${SRC_LIST}  
    @echo "Produce object files"  
    ${CC} -c $<
```

```
clean:  
    @echo "Performing cleanup"  
    -rm -vf *.o ${OUT}
```



GNU Make

Това са само начални напътствия. Има още много възможности и функционалност, описана в документацията.



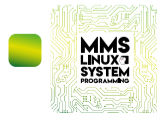
Бележки по материалите и изложението

- 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://en.wikipedia.org/wiki/Bootstrapping>
- https://en.wikipedia.org/wiki/CPU_modes
- https://en.wikipedia.org/wiki/System_on_a_chip
- https://en.wikipedia.org/wiki/Separation_of_mechanism_and_policy
- https://en.wikipedia.org/wiki/Monolithic_kernel
- <https://en.wikipedia.org/wiki/Microkernel>
- https://en.wikipedia.org/wiki/Linux_kernel
- https://en.wikipedia.org/wiki/System_call
- <https://www.gnu.org/software/make/manual/>
- <https://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/>
- <https://search.creativecommons.org/>



Благодаря Ви за вниманието!

