

Системно програмиране за Линукс

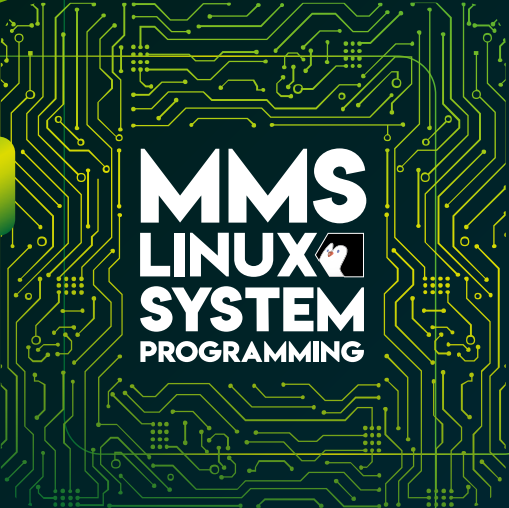

упражнения

Димитър Димитров



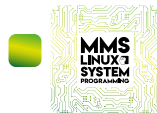
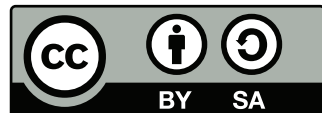
21.04.2021





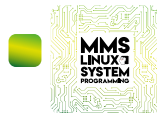
MMS LINUX SYSTEM PROGRAMMING

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



Съдържание I

- 1 Подготовка
- 2 Въведение
 - Запознаване с командния ред
 - Елементарни автоматизации с Unix Shell.
- 3 Първа Линукс програма
 - Компилатори. Фази на компилацията. ELF контейнери.
- 4 Проект
 - GNU Make. Системи за построяване на софтуер. Първи проект.
 - Системи за управление на версиите. GIT, github/gitlab.
- 5 Същински програми
 - Pthreads. Разпаралеляване на примерна задача (quicksort).
 - Механизми за синхронизация.



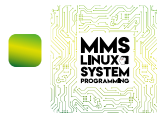
Съдържание II

- Pipes. Обмяна на съобщения между процеси.
- Mmap. Оптимизирано боравене с файлове.
- Strace. Syscalls, ioctl.

6 Вграден Линукс

- GPIO, I2C - drivers in userspace.

7 Допълнителни Материали



Подготовка

За курса ще са необходими:

- Добро владение на езика C.
- Персонален компютър или лаптоп с инсталиран Линукс.
 - Възможно е стартиране на Линукс от USB флашка, без инсталация на основния диск.
 - <https://ubuntu.com/tutorials/try-ubuntu-before-you-install>
- Резервен вариант е да се ползва Линукс чрез локална или Web Browser виртуална машина.
 - <https://ubuntu.com/appliance/vm>
 - <https://bellard.org/jslinux/vm.html?url=alpine-x86.cfg&mem=192>
- Хъс.



Относно този документ

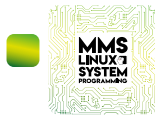
Най-новата версия може да намерите на:

<https://github.com/MM-Solutions/lsp-course>.



Терминал, shell

- Дистрибуция - що е то. Избор.
- Терминал.
- Команден ред.
 - Ctrl+D
 - Auto-complete
- Разходки из файловата система - ls, cd, pwd, tree.
- Текуща . и по-горна папка ..
- Разделител в пътя: /
- top, ps.
- su, sudo.
- root и останалите потребители



Задача

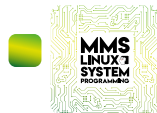
Отворете терминал и изпробвайте командите.



Текстов редактор

Пробвай и избери!

- vi ← Препоръчвам!
 - Има си самоучител: `vimtutor`
- gedit
- nano.
- emacs ← Любим редактор на множество хакери.
- Много, много други.



Изпълними файлове

- UNIX permissions: `chmod ugo+rwx MYFILE`

```
drwxr-xr-x
```

```
\. /\. /\. /
```

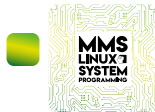
```
|  |  '-----_ Other
```

```
_ _ | _ _ '----- Group
```

```
'-----_ User _ owner
```

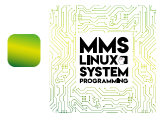
- ELF.
- Shell/Python/... scripts.
- Shebang.

```
# !/ bin / sh
```



Помощ!

- `man`
- `man 1 kill` или `man 2 kill`
 - `man man`
- `apropos`
- `whatis`
- `help` - за вградени команди на shell

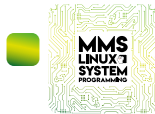


Задача

Създайте скрипт, който да:

- 1 Разпечатва един ред с текущата директория.
- 2 След това разпечатва списък с файловете.
- 3 Списъкът да съдържа освен имената на файловете, и техните размери, собственици и права.

Съвет: Ползвайте `pwd`, `man ls`



Unix shell

- Променливи на обвивката (environment variables). `printenv`
- Променливи на средата (shell variables).
- `echo`
- `PATH`
- `cat`, `echo`, `less`
- `cp`, `mv`, `mkdir`
- `ln` - soft and hard.

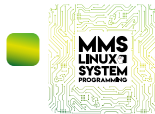


Задача

Направете мека и твърда връзка към скрипта от предишната задача.

- 1 Какъв е резултата от `ls -l`
- 2 Разпечатайте (`cat`) съдържанието на трите файла.
- 3 Изтрийте оригиналния файл. Има ли разлика в `ls -l` ?
- 4 Има ли валидно съдържание в двете връзки (`cat`) ?

Съвет: Ползвайте `pwd`, `man ls`



Многозадачност с Unix Shell

- Задачи в обвивката (shell) и процеси в OS.
- `bg`, `fg`, `&`, `wait`
- `Ctrl+Z` vs `Ctrl+C`
- `jobs`
- `kill`



Задачи

- Изпробвайте приспиването на задачи, и поставянето им във фонов режим.
 - Съвет: Ако не се сещате за други опитни зайчета, ползвайте `cat` и `top`.
- Кой е номерът на задачата, и кой е номерът на процеса (PID)?
- Изпробвайте `wait` с помощта на няколко `sleep 10` & задачи.
- Убийте един от вашите процеси. Как ще проверите дали е ”мъртъв”?



Тръби

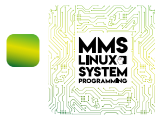
- Pipes.
- stderr, stdout, stdin.
- пренасочване чрез `>`, `>>`, `1>`, `2>`
- `tee`
- Филтри: `grep`, `sort`, `wc`, `cut`, `xxd`



Тръби - примери

Изпробвайте ги при вас!

```
$ echo "Hello ,_world"  
$ echo "Hello ,_world_1" > test.log  
$ echo "Hello ,_world_2" > test.log  
$ echo "Hello ,_world_3" >> test.log  
$ echo "Aloha!" >> test.log  
$ cat test.log  
$ cat test.log | sort  
$ cat test.log | xxd  
$ cat /bin/ls | xxd | head -10 | tee A.log  
$ cat A.log | cut -f2- -d":"
```



Тръби - примери

Изпробвайте ги при вас!

```
$ ls /bin | grep sh
```

```
$ ls /bin | grep sh | sort
```

```
$ ls /bin | sort > listing
```

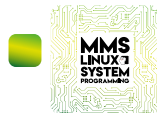
```
$ cat listing
```

```
$ ls /epa-nema-takava-direktoria | sort > listing
```

```
$ cat listing
```

```
$ ls /epa-nema-takava-direktoria 2> listing
```

```
$ cat listing
```



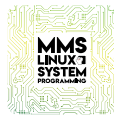
Всичко е файл!

- `/dev` : Device filesystem.
- `/proc` : Process filesystem.
- `/sys` : System filesystem.
- Даже и тръбите: `mkfifo`



Задача

- Разузнайте `dev`, `/proc` и `/sys`.
 - Съвет: Припомнете си командите от предното упражнение: `ls`, `ls -l`, `cd`, `pwd`
 - Съвет: За списък на всички файлове: `find . -type f`
 - Съвет: За търсене по част от име на файл: `find /sys -name "*temp*"`
 - Съвет: За търсене по име на файл: `find /sys -name "temp"`
- Дали `cat` работи с тези файлове?
- Какви са странните директории наименувани с числа в `/proc` ?



Задача - продължение

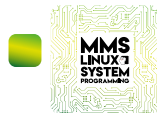
- `cat /proc/interrupts`
- `cat /proc/cpuinfo`
- `cat /proc/version`
- `ls /proc/sys/kernel/`
- `cat /sys/class/backlight/*/brightness`
- `cat /sys/class/leds/*/brightness`



Задача

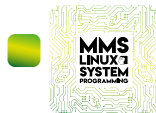
Напишете скрипт, който да разпечатва броя процесори в системата.

Съвет: Ползвайте `grep`, `wc -l`



Shell scripts

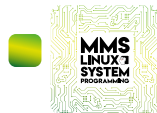
- Входни параметри - \$#, \$@, \$0, \$1, ...
- Unix SH е сложен език с много възможности които няма да разглеждаме на тези упражнения.
- За допълнително четене: <https://linuxcommand.org/tlcl.php>



CRON

Изпълнение на наши задачи в указан час.

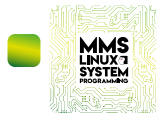
■ `man crontab`



Задача

Направете си система за мониторинг на потребителите, които ползват Вашата система. Нека всяка минута да се добавят записи в `$HOME/users.log` с информация за датата, и кои потребители ползват системата.

- Съвет: Ползвайте `w` за списък на потребителите.
- Съвет: Ползвайте `date` за датата.
- За напреднали: Филтрирайте така, че имената на потребителите да се показват само по един път.



Компилатори

- GCC vs clang
- Крос компилатори - host vs target.
- `man gcc`



- └ Първа Линукс програма
 - └ Компилатори. Фази на компилацията. ELF контейнери.

Фази

- Препроцесор.
 - Изход: С код
- С компилатор.
 - Изход: асемблерен код
- Асемблер.
 - Изход: обектен файл (ELF object).
- Свързващ редактор (linker).
 - Изход: ELF executable.

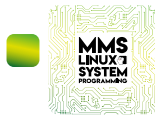


GCC - опции

- `-o OUTFILE` : в кой файл да се запише изхода.
- `-Os`, `-O0`, ... `-O3` : ниво на оптимизация.
- `-c` : да се компилира и асемблира, без свързване.
- `-g` : да се добави debug информация.
- `-Wall -Wextra` : да се предупреждава при съмнителен код.
- `-Werror` : да се излиза с грешка при съмнителен код.

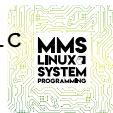
Пример:

```
$ gcc -O2 -Wall -Wextra test.c -o test  
$ file test
```



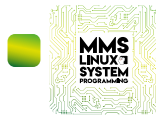
Задача

- Напишете C програма за събиране на две числа, подадени от командния ред.
- Програмата да е разделена в два C файла - за `main()` и `calc_sum()` функциите.
- Да има header с декларация на `calc_sum()`
- Напишете скрипт, който компилира програмата.
- За по-напредналите: `objdump -d calc.elf | less`
- За по-напредналите: `file calc.elf` - защо е маркиран като `dynamically linked`? Ще се промени ли нещо, ако се ползва `-static` флаг на компилатора и свързващия редактор?



GNU Make

- GNU Make има множество алтернативи, но въпреки това е все още популярна.
- Позволява фин контрол върху процеса на построяване (build).
 - Това обаче може и да се окаже недостатък, защото писането на GNU Make правила изисква задълбочено познание за почти всяка стъпка от построяването.
 - По-съвременните системи за построяване обикновено включват голям набор готови правила.
- Правилно описани правила за построяване позволяват:
 - Автоматично разпаралелване.
 - Бързи и сигурни надстроявания (incremental builds).

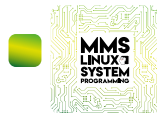


GNU Make синтаксис

■ Запомнете:

```
# This is a comment.
```

```
target: dependencies  
      commands
```



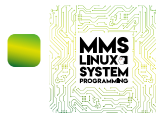
Задача

- Напишете `Makefile` за C програмата от предишното упражнение.
- За по-напредналите: Опитайте да направите своя `Makefile` по-универсален посредством променливи.
- За по-напредналите: Проследете `exit code` при успешно и при неуспешно построяване. Ползвайте специалната променлива на `sh` - `$?` .



GIT

- Най-популярната система за контрол на версиите.
- Масово използвана при разработка и поддръжка на софтуер.
- Но може да се ползва и за други цифрови проекти:
 - Документация (като тази презентация).
 - Проекти за електронни схеми и платки.
 - Проекти за механични изделия.
- Оптимизирана да работи с текст, но се справя и с двоични файлове.
- Не е система за ревю!
- Други подобни системи: hg, svn, bazaar, cvs.



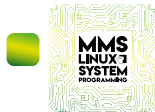
GIT сървъри

- GIT може пълноценно да работи без сървър, само с локален проект.
 - Всяко GIT дърво съдържа независимо копие на историята на проекта.
- Сървърът позволява обаче да споделяте кода си с други хора.
 - `git push` : записване на Вашата история на сървър (remote).
 - `git fetch` : сваляне на историята на даден сървър.
- Съществуват безплатни сървъри, на които да публикувате проектите си.



GIT - цели

- Управление на версиите.
- Проследяемост.
- Паралелна работа на множество разработчици.
- Пълноценност без свързаност към интернет.
- Гъвкавост при издаване (release flow).
 - Лесно разклонение (branching).



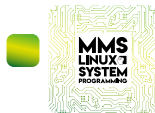
GIT - commits

- Всяка точка от историята се представя с SHA256 сума от:
 - Нейната предистория (родителските commit-и).
 - Новото състояние на дървото (разликата с родителския commit).
- Работи се в локални разклонения (branches).
- На важни точки от историята могат да се сложат етикети (tags).
- Историята може да се слива (merge commits).



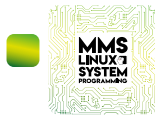
Staging

- Staging е буфер, с който контролираме кои локални промени да вкараме в следващата точка от историята (commit).
- `git add *`: добавя всички локални промени в буфера.
- `git commit`: прави нов commit с промените, които сме подготвили в staging буфера.



Документация

- <https://git-scm.com/book/>
- <https://git-scm.com/book/bg>



Файл Редактиране Изглед Помощ

v5.12-rc7 [remotes/origin](#) Linux 5.12-rc7

Merge tag 'for-5.12-rc6-tag' of git://git.kernel.org/pub/scm/linux/kernel/git/kdave/linux
 btrfs: zoned: move superblock logging zone location
 Merge tag 'locking-urgent-2021-04-11' of git://git.kernel.org/pub/scm/linux/kernel/git/tip/tip
 lockdep: Address clang -Wformat warning printing for %hd
 lockdep: Add a missing initialization hint to the "INFO: Trying to register non-static key" message
 Merge tag 'x86_urgent_for_5.12-rc7' of git://git.kernel.org/pub/scm/linux/kernel/git/tip/tip
 x86/traps: Correct exc_general_protection() and math_error() return paths
 RAS/CEC: Correct ce_add_elem()'s returned values
 Merge branch 'for-5.12-fixes' of git://git.kernel.org/pub/scm/linux/kernel/git/dennis/percpu
 percpu: make percpu_nr_empty_pop_pages per chunk type
 Merge tag 'scsi-fixes' of git://git.kernel.org/pub/scm/linux/kernel/git/jbjs/scsi
 scsi: scsi_transport_atp: Don't block target in SRP_PORT_LOST state
 scsi: target: iscsi: Fix zero tag inside a trace event
 ...

Linus Torvalds <torvalds@linux-foundation.org>
 Linus Torvalds <torvalds@linux-foundation.org>
 Naohiro Aota <naohiro.aota@wdc.com>
 Linus Torvalds <torvalds@linux-foundation.org>
 Arnd Bergmann <arnd@arndb.de>
 Tetsuo Handa <penguin-kernel@Hove.SAKURA.ne.jp>
 Linus Torvalds <torvalds@linux-foundation.org>
 Thomas Tai <thomas.tai@oracle.com>
 William Roche <william.roche@oracle.com>
 Linus Torvalds <torvalds@linux-foundation.org>
 Roman Gushchinn <guro@fb.com>
 Linus Torvalds <torvalds@linux-foundation.org>
 Martin Wilck <mwilck@suse.com>
 Roman Bolshakov <r.bolshakov@yadro.com>
 ...

2021-04-12 01:16:13
 2021-04-11 21:53:36
 2021-04-08 11:25:28
 2021-04-11 21:47:03
 2021-03-22 13:55:25
 2021-03-21 08:49:13
 2021-04-11 21:42:18
 2021-04-08 20:28:33
 2021-04-06 18:28:59
 2021-04-10 22:51:12
 2021-04-08 06:57:33
 2021-04-10 22:29:19
 2021-04-01 12:11:05
 2021-04-04 00:54:15
 ...

SHA1: [6d48b7912cc72275dc7c59ff961c8aac7ef66a92](#) ← Ред 5 / 1063

Търсене ↓ ↑ подаване съдържимо: Точно Всички полета

Търсене

Разлики ★ Стара версия ★ Нова версия Контекст в редове: 7 Празните знаци без значение |Поред|

Явотор: Arnd Bergmann <arnd@arndb.de> 2021-03-22 13:55:25
 Поглед: Ingo Molnar <mingo@kernel.org> 2021-03-22 23:07:09
 Поглед: [3a85969e9d912d5dd853e2ee37b5781266e08e77](#) (lockdep: Add a missing initialization hint to the t
 Дете: [add6b92660b3dc6a55465d8d7718b4b1338f34f0](#) (Merge tag 'locking-urgent-2021-04-11' of git://gl
 Клон: [remotes/origin/master](#)
 Сегба: [v5.12-rc3](#)
 Предества: [v5.12-rc7](#)

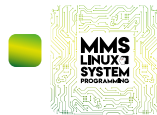
lockdep: Address clang -Wformat warning printing for %hd
 Clang doesn't like format strings that truncate a 32-bit
 value to something shorter:
 kernel/locking/lockdep.c:709:4: error: format specifies type 'short' but the argument has typ
 In this case, the warning is a slightly questionable, as it could realize
 that both class->wait_type_outer and class->wait_type_inner are in fact
 8-bit struct members, even though the result of the ? operator becomes an
 'int'.
 However, there is really no point in printing the number as a 16-bit
 'short' rather than either an 8-bit or 32-bit number, so just change
 it to a normal %d.
 Fixes: [de8f5e4f2dc1](#) ("lockdep: Introduce wait-type checks")

Кръпка Дърво
 Кментари
 kernel/locking/lockdep.c



GIT команди

```
$ git init my-new-repo  
$ git clone https://github.com/MM-Solutions/lsp-course  
$ git add *  
$ git status  
$ git commit  
$ git push  
$ git pull  
$ git log  
$ git log --decorate --graph --oneline  
$ git show COMMIT  
$ gitk
```



Задача

- Направете локален ГИТ проект и добавете С задачата от предишното упражнение в него.

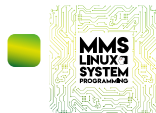
```
$ git init lsp-course-labs
```

```
$ cd lsp-course-labs
```

```
$ git add *
```

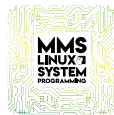
```
$ git commit
```

- Добавете README.md с кратко описание, и го публикувайте като отделен GIT commit.
- Разгледайте си историята с `git log` и `gitk`.



Задача

- Направете си публично ГИТ хранилище (GIT project, GIT repo).
- Предложения за сървър:
 - `github.com`
 - `gitlab.com`
 - `bitbucket.org`
- Следвайте съответните инструкции за настройка на достъпа. Най-често е посредством `ssh key`.
- Публикувайте си С задачата използвайки `git push`.
- Добавете С задачата от предишното упражнение към новото ГИТ хранилище, и я публикувайте.
- Споделете си новото ГИТ хранилище.



Задача

- За по-напредналите: свалете голям проект (например <https://github.com/torvalds/linux>), и изследвайте историята. Изпробвайте командите:

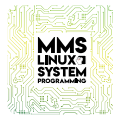
```
$ git log
```

```
$ git log --decorate --graph --oneline
```

```
$ gitk
```

```
$ git show COMMIT
```

- Прегледайте няколко произволни commit-a.
 - Commit message разбираем ли е, дори за „новодошъл“?



PThreads

- Posix Threads е стандартна библиотека за нишки.
- Документация: `man pthreads`
- Работи и за отделни функции: `man pthread_create`



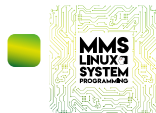
Често използвани функции

- `pthread_create`
- `pthread_join`
- `pthread_mutex_lock`
- `pthread_mutex_unlock`



Задача

- Преправете задачата от предишното упражнение да смята сумата в една отделна нишка.
- Помислете как ще върнете резултата от пресмятането така, че да не се вика `printf` вътре в новата нишка.
- Направете нов `GIT commit` за промяната.



Задача за напреднали

- Напишете програма, която създава отделна нишка за всяко `/dev/input/event*` устройство.
- Всяка нишка да чете и декодира събитията от съответното у-во.
- Изполвайте функцията `do_capture` от <https://github.com/freedesktop-unofficial-mirror/evtest/blob/master/evtest.c>
- Споделете в github.
- Възможно ли е тази програма да се напише без pthreads?



Критична секция

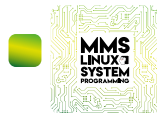
- Пуснете програма
`listings/lab_synchronization/01-parallel-counting.`
- Каква е грешката в тази програма? Поправете я.
- Програмата дава пример какво е критична секция.



Критична секция - анализ

- Измерете времето за изпълнение на програмата с и без поправката.
- Колко е голяма разликата на Вашата машина?
- Защо има разлика в бързодействието?

```
$ time ./main
```



Мъртва хватка

- Пуснете програмата
listings/lab_synchronization/02-counting-deadlock.
- Защо не приключва?

```
$ gdb ./main
```

```
run
```

```
Ctrl+C
```

```
info threads
```

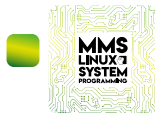
```
thread 2
```

```
backtrace
```

```
up
```

```
up
```

```
list
```



Pipes

- Не само UNIX Shell може да създава тръби.
- Защо обаче `man 2 pipe2` не приема аргумент за команда, която да изпълни?
 - Защото създаването на нов процес е отделен syscall: `man 2 fork`
 - Защото изпълнението на нова програма е отделен syscall: `man 3 exec1p`
 - Защото задаването на `stdout` и `stdin` са под контрола на процеса: `man 2 dup2`



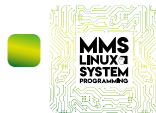
Задача 1

- Преправете задачата от предишното упражнение да смята сумата от две числа използвайки външна програма.
- Съвет: `echo "2+3" | bc`
- Препоръчителна последователност:
 - Създайте тръба към `stdin` на `bc` използвайки `man` `popen`.
 - Запишете израза за смятане като данни във върнатия `FILE *`.
- Забележете, че пренасочихме `stdin` на `bc` към `FILE *` на нашия процес.
- `stdout` на `bc` се наследява от `parent` процеса, и е изхода на терминала.
- Публикувайте като нова директория във Вашето `GIT` хранилище.



Задача 2 - подготовка

- Извикайте външна програма, например `/bin/ls`, от ваша C програма.
- Препоръчителна последователност:
 - Създайте нов процес с `man 2 fork`.
 - В новия child процес изпълнете `/bin/ls` използвайки `man execvp`.



Задача 2 с повишена трудност

- Преправете задачата от предишното упражнение да смята сумата от две числа използвайки външна програма.
- Прихванете както `stdin`, така и `stdout` на външната програма.
- Съвет: Ще се наложи да направите две тръби към и от `bc`.
- Препоръчителна последователност:
 - Създайте две тръби с `man 2 pipe2`. Една, за да се свържете със стандартния вход на `bc`, и една за да се свържете със стандартния изход на `bc`.
 - Създайте нов процес с `man 2 fork`.
 - Задайте вход/изход на `child` процеса да са `file descriptors` от тръбата. Използвайте `man stdout`, `man 2 dup2`



Задача 2 с повишена трудност

- Трябва да сте получили 4 file descriptors. По два за всяка тръба.
- В child процеса сменете stdin/stdout с файлови дескриптори от двете тръби. Пример:

```
int P1[2];
```

```
pipe(P1);
```

```
....
```

```
/* Replace stdin with P1[0] for current process. */
```

```
dup2(P1[0], STDIN_FILENO);
```

- Публикувайте като нова директория във Вашето GIT хранилище.

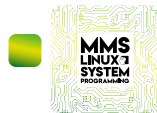


mmap

- Как данни от диска се появяват в оперативната памет на процеса?
- Същият механизъм се използва за зареждане на динамични библиотеки:

```
$ cat /proc/self/maps
```

- `man 2 mmap`



Задача 1

- Реализирайте програма за разпечатка на текстов файл, подобно на `cat`.
- Използвайте `man 2 mmap` за достъп чрез указател до файла.
- Използвайте `man 2 fstat` за да узнаете колко е голям файла.
- Публикувайте като нова директория във Вашето GIT хранилище.



Задача 1 - допълнение

- Разпечатайте адреса, върнат от `mmap` посредством `printf`.
- Забавете програмата с `usleep` или `getchar`.
- Сравнете адреса с `cat /proc/PID/maps`

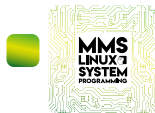


Задача 2

- Реализирайте двоично търсене на дума в речник.
- Използвайте файла `/usr/share/dict/words` за входни данни.
- Публикувайте като нова директория във Вашето GIT хранилище.

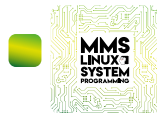


- Изпълнете задачата от упражнението с `make` използвайки `strace`.
- Защо дори и за елементарна "Hello, world!" програма се извиква `mmap`?
- Компилирайте същата програма със `-static` параметър на свързващия редактор. Все още ли се извиква `mmap`?



strace

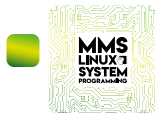
- Изпълнете задачата от упражнението с `mmap` използвайки `strace`.
- Изпълнете задачата от упражнението с `pipe2` използвайки `strace -f`.
- Отбележете кои `syscalls` използват вашите програми.



strace

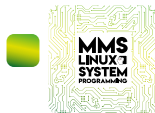
- `strace` може да се използва за изследване при липса на изходен код.
- Изпълнете следните команди, и си припомнете упражнението с `/proc`.

```
# List all processes , and write
# strace output (stderr) to out.txt
$ ps aux 2>out.txt
# Check which files were accessed
$ grep '^open' out.txt
```



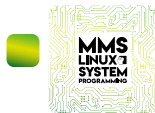
ioctl

- `ioctl` е механизъм за настройване на параметри на device файлове или други специални файлове.
- `ioctl` приема като аргумент файлов дескриптор.
- `man ioctl`
- `man ioctl_list`



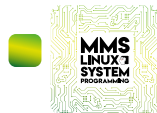
Задача 2

- Реализирайте програма за четене размерите на текущия терминал.
- `man 2 ioctl_tty`.
- Изпълнете `ioctl TIOCGWINSZ` върху файл `/dev/tty`.
- Изпробвайте с различни размери на Вашия терминален прозорец.
- Публикувайте като нова директория във Вашето GIT хранилище.



ioctl/strace

- Проследете програмата с `ioctl` използвайки `strace`.
- Открийте добавения от Вас `ioctl` в изхода.



Стандартни интерфейси към периферията

■ TODO.



Други курсове

- https://man7.org/training/download/Linux_System_Programming-man7.org-mkerrisk-NDC-TechTown-2020.pdf
- <https://bootlin.com/training/>
- <https://github.com/e-ale>

