Програмиране в UNIX среда

Файлова система, команди за четене и навигация, работа с файлове и търсене. Графична среда.

Файлови системи.

Файлова система

- В изчислителната техника **файловата система (file system)** е метод за съхранение и организация на компютърни файлове и данните в тях с цел по-лесно търсене и достъп.

- Файловата система може да ползва устройство за съхранение на данни, като например **твърд диск или CD-ROM** и осъществява физическото локализиране на файловете. Може също така да предоставя достъп до данните от сървър, като самата тя действа като клиент за мрежов протокол (например **NFS, SMB, или 9P**

Видове файлови системи

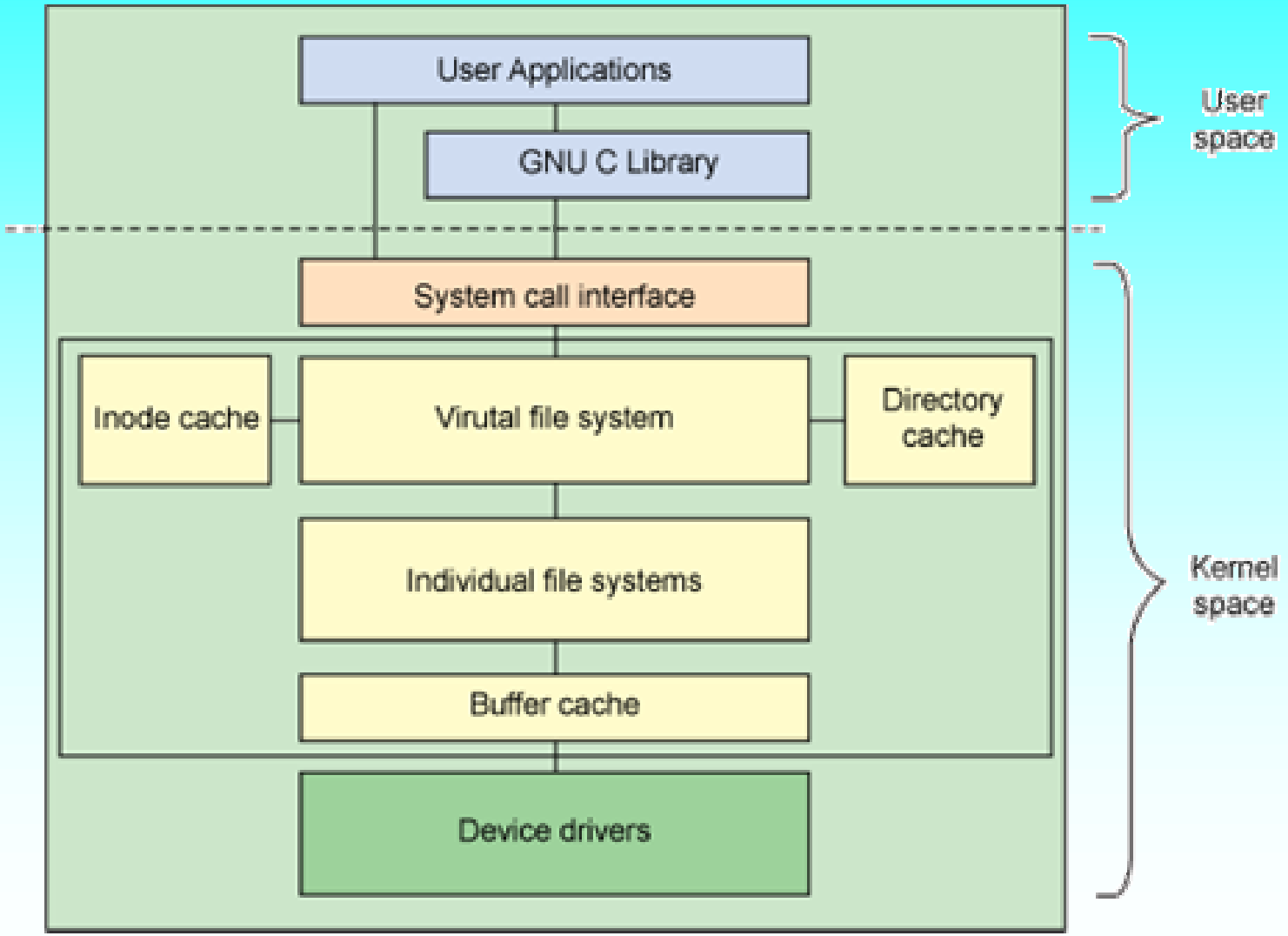
- **Дискови файлови системи** (FAT, VFAT, ext2, ext3, ISO9660, NTFS, UFS и много други)
- **Разпределени файлови системи** (9P, AFS, NFS, SMB и много други)
- **Псевдо- и виртуални файлови системи** (devfs, procfs, specfs, sysfs)
- **Криптирани файлови системи** (SSHFS, eCryptfs, EFS и други)

Виртуална файлова система (Linux)

- **Виртуалната файлова система (**virtual file system - VFS**)** е абстрактна надстройка над конкретните файлови системи. Целта на VFS е да осигури на програмните приложения еднотипен достъп до различни конкретни файлови системи. VFS може например да се използва за достъп до локални или мрежови устройства за запис на информация по прозрачен начин без потребителската програма да забележи каквато и да е разлика.

- **VFS** може да се използва за осъществяване на връзка между различни файлови системи на различни операционни системи (например **Windows, Mac OS и Unix**), така, че приложението да има достъп до техните локални файлови системи, без да се нуждае от допълнителна информация за файловата система на която се намират данните.

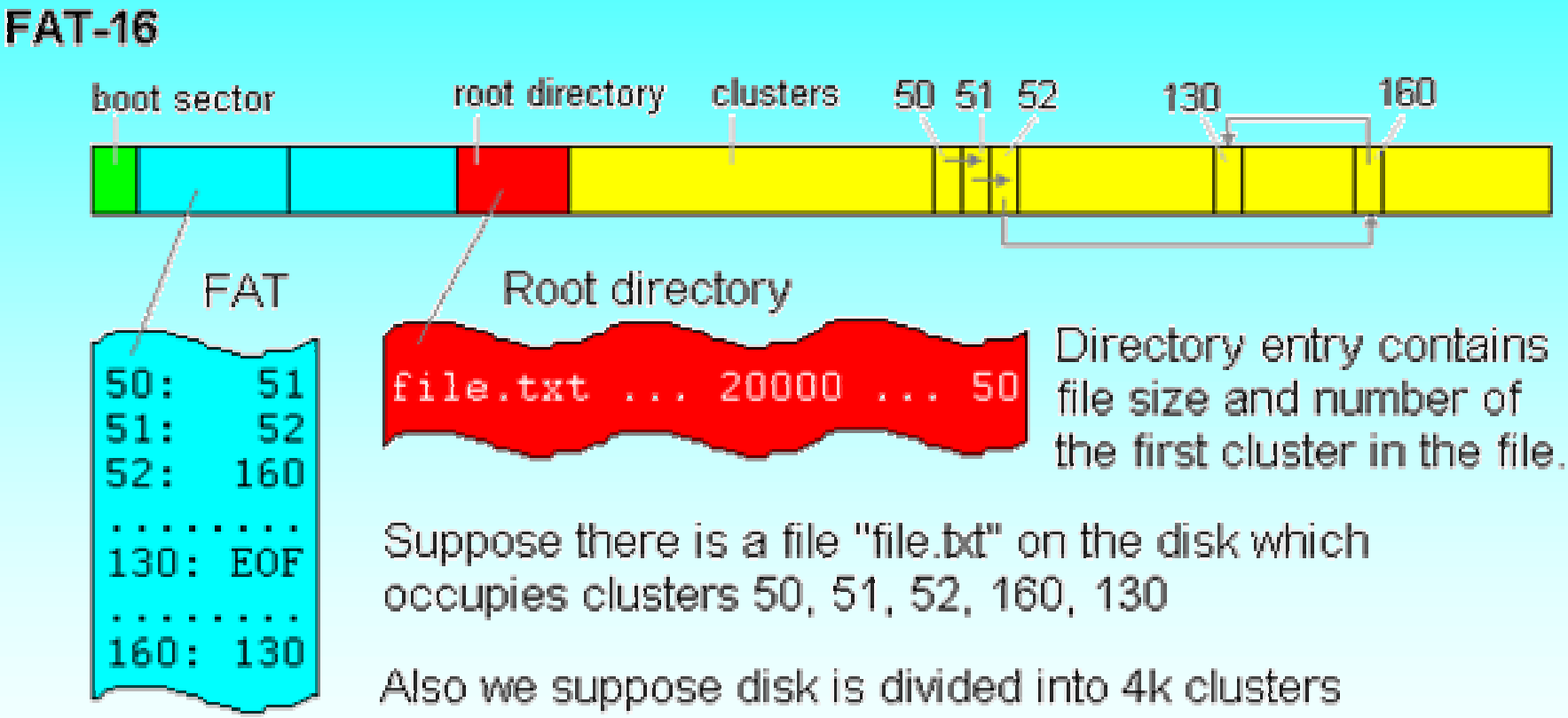
Виртуална файлова система (Linux)



FAT (File Allocation Table)
FAT12, FAT16, FAT32, exFAT (FAT64)

- **FAT е** е файлова система Microsoft за MS-DOS и е основна файлова система за Microsoft Windows до версия Windows Me включително. FAT е частично патентована.
- FAT е сравнително проста и се поддържа практически от всички операционни системи за PC. Това я прави идеален формат за дискети и твърдотелни памети, както и за споделяне на данни между операционните системи на “dual boot” PC.
- Най-разпространената реализация има сериозен недостатък – когато се трият файлове и се записват нови файлове на носителя, фрагментите на директориите се “разхвърлят” по целия диск, което прави писането и четенето от диска бавен процес. **Дефрагментацията “defragmentation”** е решение на този проблем, но тя самата е бавна и за да има ефект трябва да се извършва регулярно. Твърдотелни носители не е се дефрагментират.

Организация на FAT файлова с-ма



NTFS
(1993 г.)

- При **NTFS** всички данни за файла- име, време на създаване, права за достъп и самото съдържание на файла се съхраняват като метаданни. Този абстрактен подход позволява лесно добавяне на нови възможности на файловата система по време на развитието на **Windows NT**. Интересен пример е добавянето на поле за индексация, ползвано от софтуера на **Active Directory**.

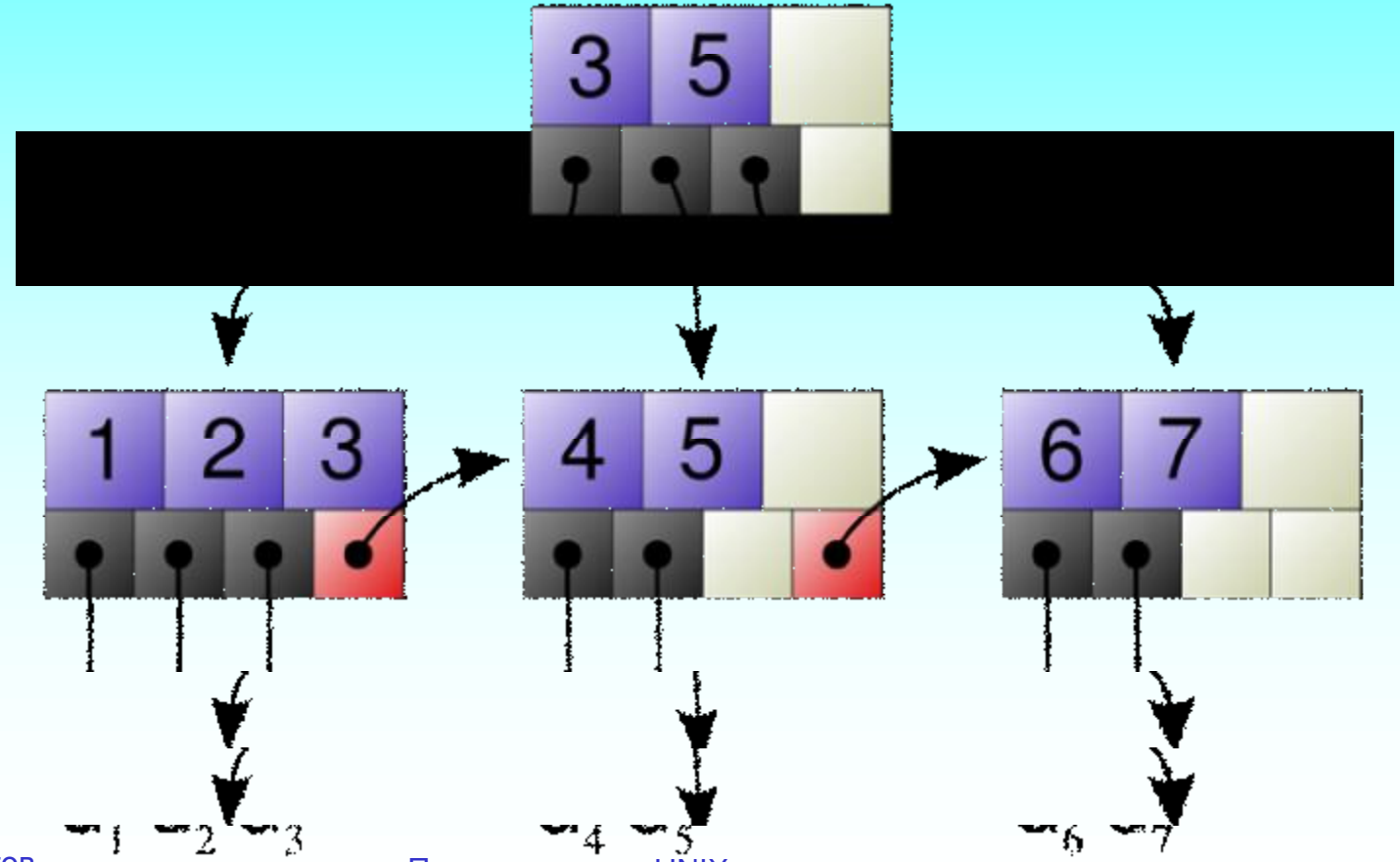
- NTFS разрешава да се използват за имена на файлове и др. произволна последователност от 16-битови стойности, т.е. поддържа се UTF-16 кодировка.

- NTFS използва **B+ дървета (B+ trees),** за да индексира данните на файловата система. Въпреки, че е сложно да се реализират, те позволяват по-бързо търсене на файловете. Използва се журнал на файловата система, за да се осигури цялост на файловата система, но не и на съдържанието на отделните файлове. Системи използващи NTFS са по-надеждни от тези ползващи FAT.

- **Master File Table (MFT)** съдържа метаданни за всеки файл, директория и метафайл на NTFS. Тя съдържа имена на файлове, локация, размер и права за достъп. Нейната структура поддържа алгоритми, които минимизират фрагментацията на диска. Записът на директориите съдържа името на файла и файлов идентификатор ("file ID“), който е представя файла в MFT.

B+ дърво

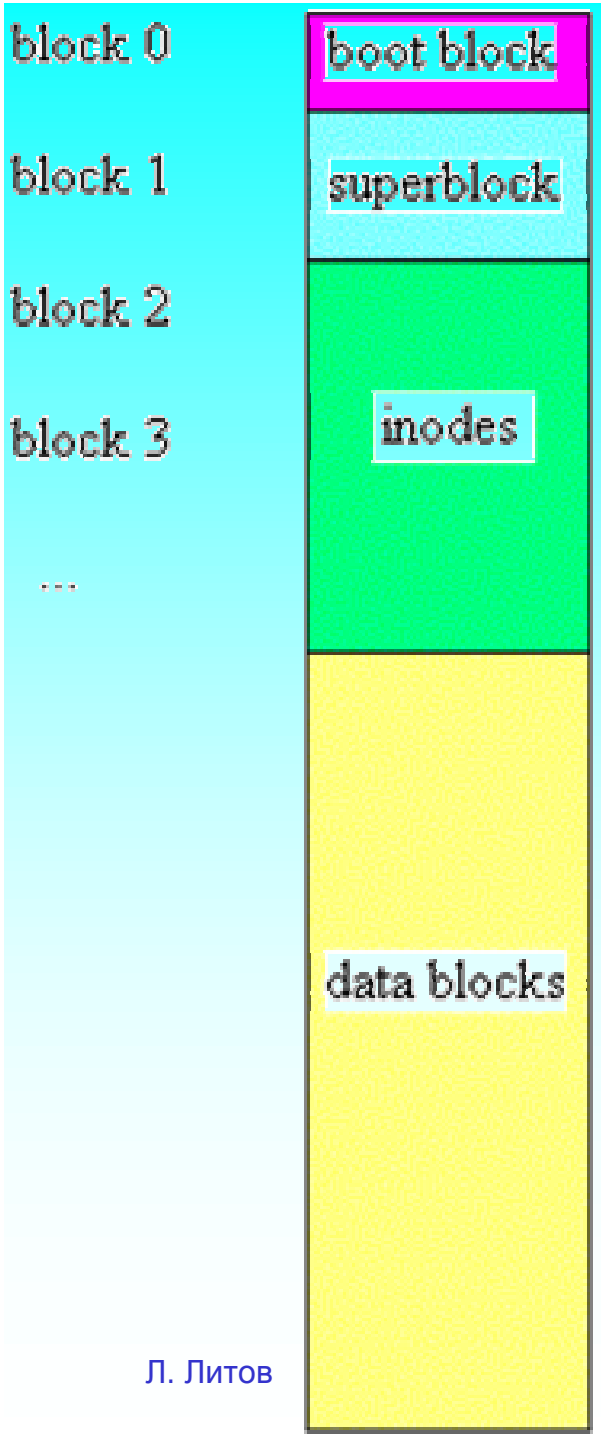
Ø **B+ tree (Quaternary Tree)** е дърво, което представя сортирани данни по начин, който позволява ефективно добавяне, получаване и триене на записи, всеки от които се идентифицира с ключ.



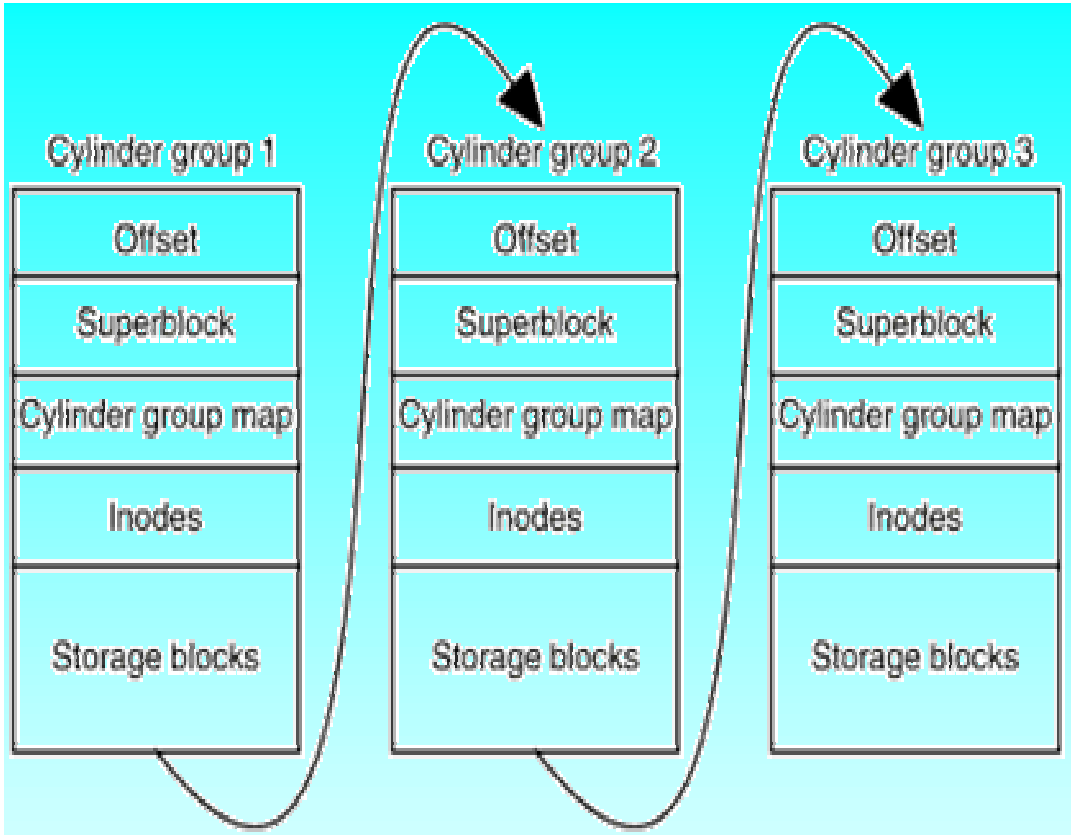
UFS (Unix File System)

- Няколко **блока** в началото на **дяла (partition)** резервирани за **boot** blocks, които трябва да се инициализират отделно от файловата с-ма
- **Суперблок (superblock)**, съдържащ магично число (magic number) показващо, че това е **UFS** файловата система и други важни числа, описващи геометрията на файловата система, а също статистика и параметри на оптимизация.
- Колекция от групи цилиндри. Всяка група цилиндри има следните компоненти:
  - "backup" копие на суперблока
  - "header" на групата от цилиндри, със статистика и т.н., подобни на тези на суперблока
  - **i-възли (inode)**, всеки съдържащ атрибути на файлове
  - Блокове с данни (data blocks).
- I-възлите са номерирани последователно. Първите няколко i-възела са резервирани по исторически причини, последвани от i-възлите за root директорията.
- Файловете на директориите съдържат само списък на имената на файловете в директорията и i-възлите асоциирани към всеки файл. Всички метаданни на файла са пазят в i-възела.

UFS (Unix File System)



- Няколко блока в началото на дяла са за boot. Те се инициализират отделно от файловата с-ма
- Следва суперблок (superblock). Той съдържа магично число (magic number) показващо, че това е **UFS** и други важни числа, описващи геометрията на файловата система + статистика + параметри на оптимизация.

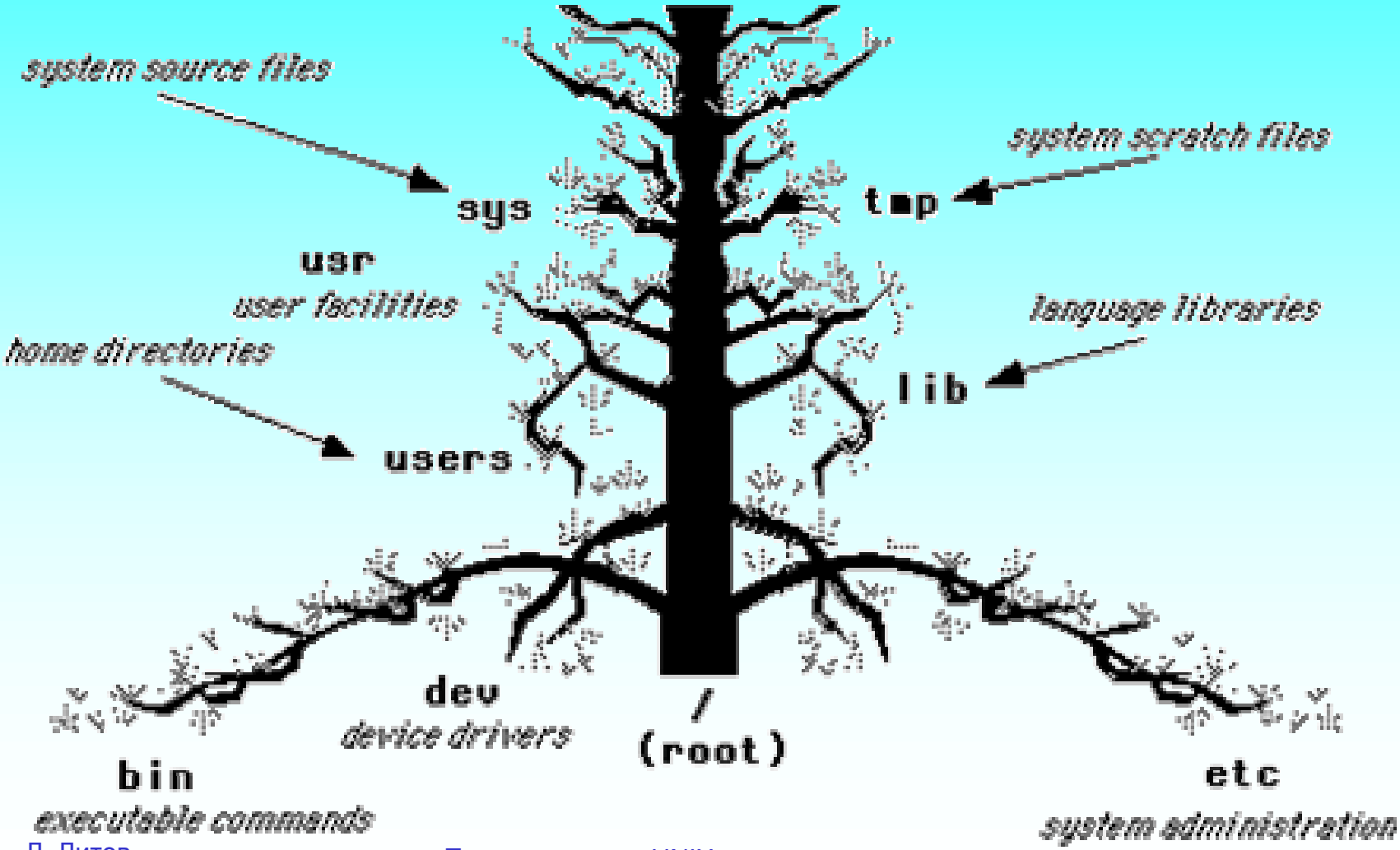


UFS (Unix File System)

- Следва колекция от групи цилиндри. Всяка група цилиндри има следните компоненти:
  - “backup” копие на суперблока
  - “header” на групата от цилиндри, със статистика и т.н., подобни на тези на суперблока
  - i-възли (inode), всеки съдържащ атрибути на файлове
  - Блокове с данни (data blocks).

- I-възлите са номерирани последователно. Първите няколко i-възела са резервирани по исторически причини, последвани от i-възлите за root директорията.
- Файловете на директориите съдържат само списък на имената на файловете в директорията и i-възлите асоциирани към всеки файл. Всички метаданни на файла са пазят в i-възела.

UFS структура



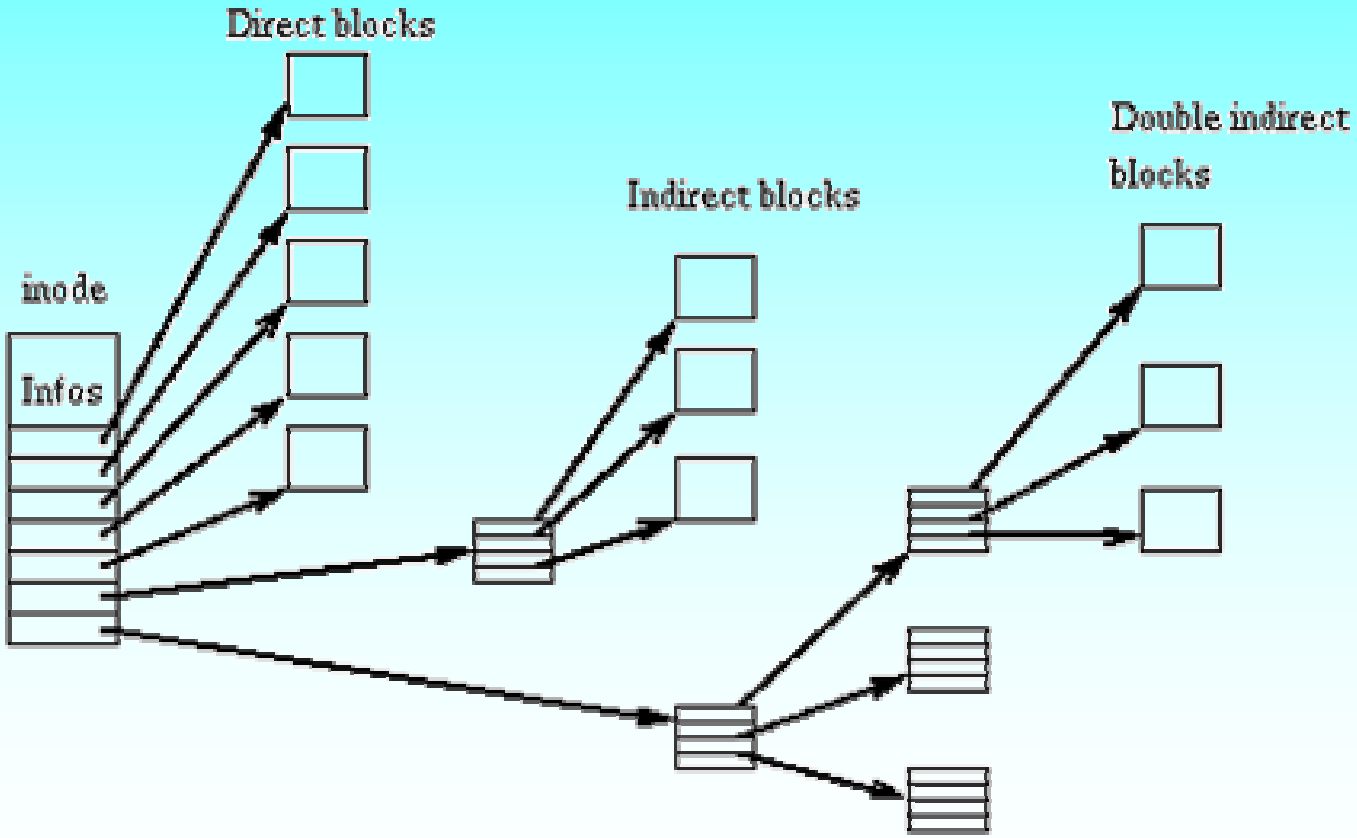
ext2 (Linux)

- **ext2 (second extended file system)** е файлова с-ма за **Linux** ядро. Разработена е от **Rémy Card** с цел замяна на ext. Достатъчно бърза е и се използва за **“benchmark” стандарт**.

- **“Каноничната” реализация на ext2 е ext2fs драйверът за файлова с-ма в ядрото на Linux.** Други реализации (с различно качество и пълнота) съществуват в **GNU Hurd, Mac OS X, Darwin, някои BSD ядра, има и драйвери за Microsoft Windows.** До появата на ext3, ext2 е била файлова система по подразбиране в няколко Linux дистрибуции, включително **Debian и Red Hat Linux.**

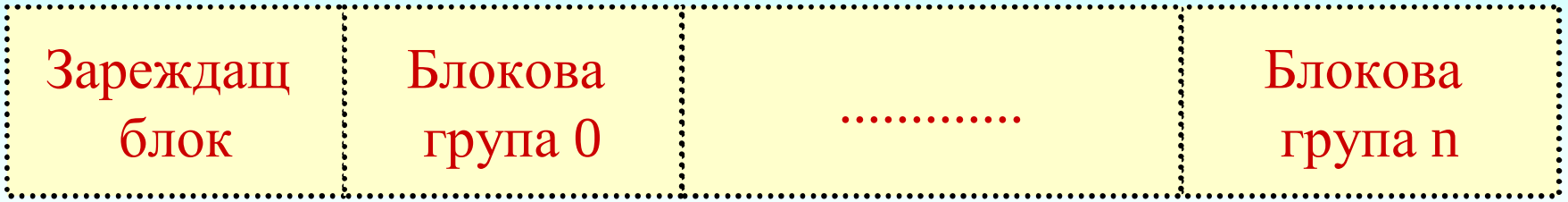
ext2

- Данните се адресират от **15 указатели** - първите **12 директно** сочат към първите 12 блока от данни, а останалите **3 са индиректни указатели.**



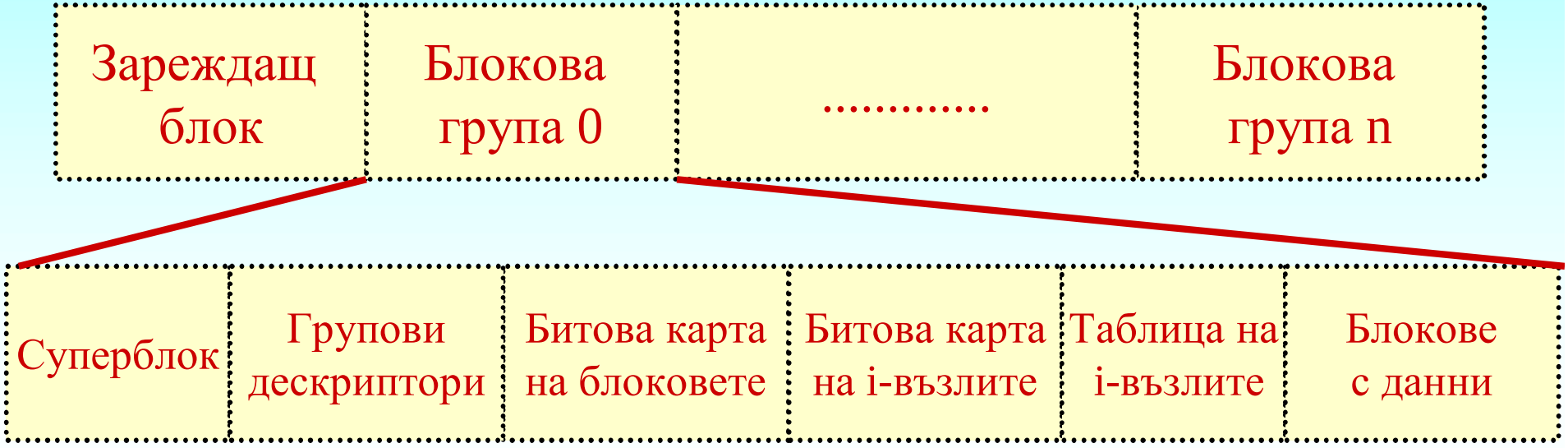
ext2

Ø Пространството в **ext2** е разделено на блокове, които са организирани в **групи от блокове**, аналогични на **групите от цилиндри в UFS**. Това е направено с цел да се избегне фрагментацията и да се намали търсенето по диска, когато се чете голямо количество от последователно записани данни.



ext2

- Всяка група от блокове се състои от:
- **суперблок**
- **групови дескриптори**
- **битова карта (bitmap) на блоковата група**
- **битова карта на i-възлите**
- **таблица на i-възлите**
- **следват блоковете с данните**



ext2

- **Суперблокът** съдържа важна информация, която е **критична за файловата с-ма** (брой на блоковете и на i-възлите) за това негово backup копие се прави в някои от блоковете на групата.

- **Груповият дескриптор** съдържа блокови номера, отговарящи на местоположението на битовите карти на блоковете, битовите карта на i-възлите и на таблицата на i-възлите.

- **Таблицата на i-възлите** е с фиксиран размер и съдържа по един елемент за всеки i-възел в блоковата група. Индексните възли (i-възлите) сочат към данните на файловете.

- **Битова карта за разпределение на i-възлите** показва използването им в блока. Всеки бит в битовата карта кореспондира с елемент на таблицата на i-възлите - когато има записан файл битът се установява в “1”, а когато няма – “0”.

- **Битовата карта за разпределение на блоковете** се изгражда по аналогичен начин.

ext3

- **ext3** е **журнална файлова система** и е надстройка на **ext2** със средства за протоколиране на измененията на файловата с-ма в журнален файл.
- Преминаването от **ext2** към **ext3** може да стане по време на работа с помощта на програми + командата tune2fs.
- **ext2** може да работи в три режима.

Режими на работа на ext3

- **Пълно протоколиране (full data journaling).** Протоколират се измененията в данните и метаданните. Всички изменения се записват най-напред върху журналния файл, а след това върху тяхното местоположение. При авария, с помощта на журналния файл могат да се възстановят данните и метаданните. **Това е най-надеждният подход, но и най-бавният.**

- **Протоколиране с обратен запис (writeback).** Протоколират се промените само в метаданните. **Най-бързият метод, но са възможни повреди в данните при авария.**

- **Подредено протоколиране (ordered).** Протоколират се само метаданните, но се гарантира, че съдържанието на файла се записва на диска преди съответните метаданни да се променят в журнала.

proc файлова с-ма

- **/proc** файловата с-ма е създадена, за да представя **информация в реално време за статуса на ядрото** и процесите в системата. Тя се създава от ядрото по време на изпълнение.
- Чрез нея потребителят може да получи **детайлна информация за с-мата** – от състоянието на апаратните средства до трафика на мрежата.
- **/proc** файловата с-ма **съществува само в паметта** и нейните файлове не са записани на носител – те дават на потребителя точка за достъп до информация на ядрото ,която се генерира при поискване.

devfs

- **devfs** е специализирана файлова с-ма, използвана да представя **файлове на устройства (device files)**, т.е. абстракция за достъп към I/O устройства и друга периферия.
- **device file** е **интерфейс за драйверите на устройствата**, който се вижда във файловата с-ма, като обикновен файл. Това позволява софтуерът да взаимодейства с драйверите на устройствата използвайки стандартните входно-изходни системни извиквания, което опростява много задачи.
- **Device file** често предоставят прост интерфейс към периферни устройства, като например принтери. Но те могат да бъдат използвани за достъп до специфични ресурси на съответните устройства, като например дискови дялове (disk partitions).
- **device files** се използват за достъп до системни ресурси, които нямат връзка към конкретно устройство – **/dev/null, /dev/zero, /dev/random** (генератори на случайни числа)

Команди за четене и навигация, работа с файлове и търсене.

Потребителски интерфейс на UNIX: **ОБВИВКА (SHELL)**

- Предоставя **команден ред (command line)**, като интерфейс между потребителя и системата
- Обвивката е просто програма, която се стартира автоматично, когато се “логнете”
- Използва команден език
  - Позволява програмиране (shell scripting) в обкръжението на обвивката
    - Използва променливи, цикли, условия и т.н.
  - Приема команди и често извършва **системни извиквания (*system calls*)**, за да ги извърши

Различни UNIX обвивки

- sh (Bourne shell)
- ksh (Korn shell)
- csh (C shell)
- tcsh
- bash
- Различават се най-вече в детайлите

Login scripts

- Не е необходимо всеки път да се въвеждат псевдоними (aliases), променливи на средата (environment variables) и т.н.
- Всички тези неща могат да се направят в скрипт, който се изпълнява всеки път, когато се стартира обвивката
- За **ksh**:
  - `~/.profile` – се прочита от login shell
  - `~/.kshrc`
- За **tcsh**:
  - ~/.login
  - ~/.cshrc

Пример .profile (само част)

```
# set ENV to a file invoked each time sh is started for interactive use.
ENV=$HOME/.shrc; export ENV
HOSTNAME=`hostname`; export HOSTNAME
PS1="$USER@$HOSTNAME>"

alias 'll'='ls -l'
alias 'la'='ls -la'
alias 'ls'='ls -F'
alias 'rm'='rm -i'
alias 'm'='more'

set -o vi
echo ".profile was read"
```

stdin, stdout, и stderr

- Всяка обвивка (и всяка програма) **автоматично отваря три "файла"**, когато стартира:
  - **Стандартен вход (Standard input) stdin**: обикновено за клавиатурата
  - **Стандартен изход (Standard output) stdout**: обикновено към терминала
  - **Стандартна грешка (Standard error) stderr**: обикновено към терминала
- Програмите ползват тези файлове, когато четат, пишат и дават съобщение за грешка.

Пренасочване на stdout

- Може да се накара програмата вместо да пише на терминала (екрана) да пише в друг файл. Това се прави, чрез > оператора
- >> оператора се използва за добавяне към файл
- Примери:
  - `man ls > ls_help.txt`
  - `Echo $PWD > current_directory`
  - `cat file1 >> file2`

Пренасочване на stderr

- Вместо да пише грешките на екрана, вие може да накарате програмата да ги пише във файл използвайки следният оператор:
  - ksh: 2> operator
  - tcsh: >& operator
- Пример (предполагаме, че j е файл, който не съществува)
  - {ajax} ls j
  - ls: j: No such file or directory
  - {ajax} ls j >& hello.txt
  - {ajax} cat hello.txt
  - ls: j: No such file or directory

Пренасочване на stdin

- Вместо програмата да чете данни от терминала, вие може да я накарате да ги чете от файл с оператора <
- Примери:
  - `Mail user@domain.com < message`
  - `interactive_program < command_list`

Програмни канали (pipes) и филтри

- **Програмен канал (pipe)**: начин да изпратите изхода на една програма към входа на друга

- **Филтър** : програма, която трансформира по някакъв начин входните данни
  - `wc` – дава броя на думите/редовете/символите
  - `grep` – търси за ред съдържащ зададен символен низ
  - `more`
  - `sort` – сортира редовете по азбучен или по числен ред

Примери на филтриране

- `ls -la | more`
- `cat file | wc`
- `man ksh | grep history`
- `ls -l | grep dkl | wc`
- `who | sort > current_users`

UNIX файлова с-ма

- Файловата с-ма е вашият интерфейс към:
  - дисковете на машината
  - дисковете на друга машина
  - изходни устройства
  - други
- *Всичко* в UNIX е файл (програми, текст, периферни устройства, терминали, …)

Работна директория

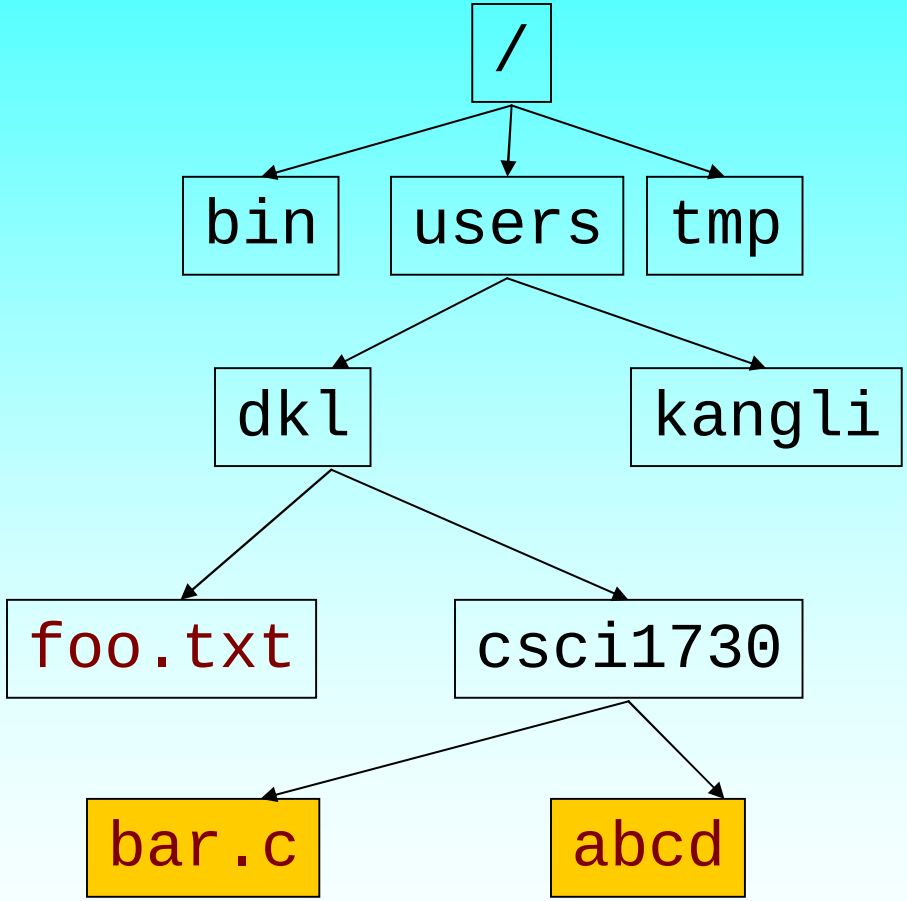
- Текущата директория в която работите
- **pwd** команда: дава пълният (абсолютният) път на вашата работна директория
- Ако не зададете друга директория, командите ще предполагат, че искате да извършвате действията в работната директория

Home (домашна) директория

- Специфично място за всеки потребител, където той си съхранява файловете
- Когато се логнете, ващата работна директория ще бъде вашата home директория
- Вашата home директория се представя от символа **~** (тилда)
- home директорията на потребител “user1” се представя с **`~user1`**

UNIX файлова йерархия

- Директориите могат да съдържат файлове или други директории
- Дървовидна структура на файловата с-ма
- **Root (коренна)** директория : **/**



Пътища (Path names)

- Разделяйте директориите с /
- Абсолютен път
  - започвате от корена и следвате дървото
  - **/users/dkl/foo.txt**
- Относителен път
  - начало – работната директория
  - .. означава за предишно ниво;
  - . означава текуща директория

- Ако /users/dkl/csci1730 е работната директория следните пътища водят към един и същи файл:
  - ../foo.txt ~/foo.txt ~dkl/foo.txt

Видове файлове

- Обикновен (`-` в първото поле)
  - Повечето файлове
  - Включително бинарни (двоични) и текстови файлове
- Директория (`d`)
  - Директорията всъщност е файл, които сочи към група от файлове
- Връзка “Link” (`l`): указател към друг файл или директория
- Специален: например периферни устройства

Създаване на връзки

- `ln -s <curr_file> <link_name>`
- тази команда създава символична връзка
- Файлът “link_name” ще бъде указател към “curr_file”, което даже може да бъде в друга директория или на друга машина

Права за достъп

- **Четене “Read” (r) 4, писане “write” (w) 2, и изпълнение “execute” (x) 1**
- За собственик, група и света (т.е. за всички)
- **chmod <mode> <file(s)>**
  - **chmod 700 file.txt (само собственикът може да чете, пише и изпълнява)**
  - **chmod g+rw file.txt**

Преглеждане на съдържанието на файл

- **`cat <filename(s)>`**
  - "concatenate"
  - Печати наведнъж целия файл
- **`more <filename(s)>`**
  - Пачати екран след екран
  - Позволява да се ходи нагоре и надолу из файла

Помощ за UNIX командите

- **`man <command_name>`** показва документацията за командата
- **`apropos <keyword>`** показва всички команди с keyword в тяхното описание

The UNIX System

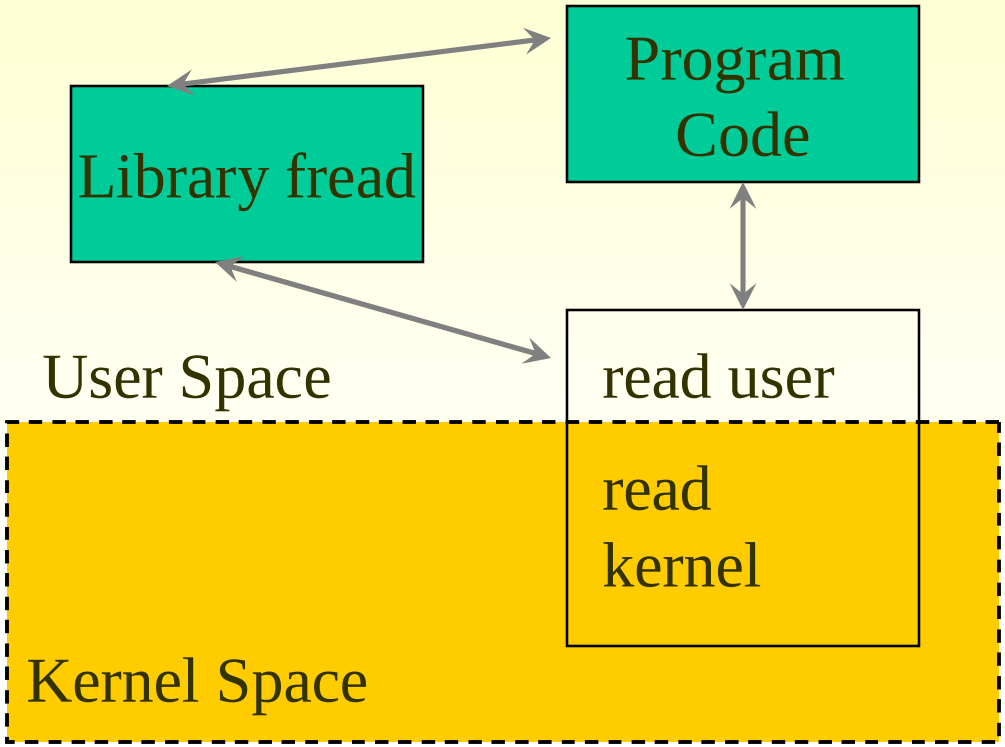
- Ядро – Сърцето на операционната с-ма
  - Process scheduling
  - I/O control (accesses)
- Обвивка (Shell) – посредник между потребителя и ядрото
- Инструменти и приложения
  - Достъпни от обвивката
  - Могат да бъдат изпълнявани независимо от обвивката

UNIX системно програмиране

- Програмите извършват *системно извикване*
- Системното извикване е процедура, която се изпълнява от операционната с-ма
- Типове системни извиквания
  - **Файл I/O**
  - **Управление на процеси (Process management)**
  - **Комуникация между процеси “Inter-process communication” (IPC)**
  - **Обработка на сигнали (Signal handling)**

Системни повиквания

- Системни повиквания
  - Интерфейс към ядрото



Основни файлови I/O

- Процесите пазят списък на отворените файлове
- Файловете могат да се отварят за четене и запис
- Всеки файл се указва от дескриптор на файла *file descriptor* (integer)
- Три файла се отварят автоматично:
  - **FD 0: стандартен вход**
  - **FD 1: стандартен изход**
  - **FD 2: стандартна грешка**

Файлови I/O системни извиквания: `open()`

- **`fd = open(path, flags, mode)`**
- път (path): символен низ, абсолютен или относителен път
- флагове (flags):
  - O_RDONLY – отваряне за четене
  - O_WRONLY – отваряне за писане
  - O_RDWR – оваряне за четене и писане
  - O_CREAT – създаване на файл, ако не съществува
  - O_TRUNC – презапис на файла, ако съществува
  - O_APPEND – пиши само в края на файла
- режим (mode): указване на права, ако се използва O_CREAT

Файлово I/O системно повикване: `close()`

- **`retval = close(fd)`**
- Затваряне на отворен файл или дескриптор
- Връща 0 при успех и -1 при грешка

Файлово I/O системно повикване: `read()`

- **`bytes_read = read(fd, buffer, count)`**
- Чете до count байта от файла и ги слага в буфер buffer
- fd: файлов дескриптор
- buffer: указател към масив
- count: брой на байтовете, които трябва да се прочетат
- Връща броя на прочетените байтове или -1 при грешка

Файлово I/O системно извикване: `write()`

- **`bytes_written = write(fd, buffer, count)`**
- Записва count на брой байта от буфера buffer във файла
- fd: дескриптор на файла
- buffer: указател към масив
- count: брой байтове за запис
- Връща броя на записани байтове или -1 при грешка

Системно извикване: `lseek()`

- **`retval = lseek(fd, offset, whence)`**
- Премества указателя на файла в ново положение
- fd: дескриптор на файла
- offset: отместване в брой байтове
- whence:
  - SEEK_SET – отместване от началото на файла
  - SEEK_CUR – отместване от сегашната позиция
  - SEEK_END – отместване от края на файла
- Връща отместването от началото на файла или -1

UNIX File access primitives

- open – отваря за четене писане или създава празен файл
- creat – създава празен файл
- close – затваря файл
- read – чете от файл
- write – пише във файл
- lseek – ходи на определен байт във файла
- unlink – трие файла
- remove – трие файла
- fcntl – контролира атрибутите свързани с файла

File I/O using FILEs

- Повечето UNIX програми ползват I/O функции от високо ниво
  - `fopen()`
  - `fclose()`
  - `fread()`
  - `fwrite()`
  - `fseek()`
- Те ползват **FILE** типът данни (datatype) вместо дескрипторите на файлове
- Необходимо е да се включи `stdio.h`

Използване на типове данни с файлов I/O

- Всички функции до сега използваха първични байтове за файлов I/O, но данните за програмите са в накакъв смислен тип (**int, char, float**, etc.)
- **`fprintf(), fputs(), fputc()` – се използват за писане във файл**
- **`fscanf(), fgets(), fgetc()` – се използват за четене от файл**

Литература:

- http://www.wylug.org.uk/talks/2003/04/unix.pdf
- http://ce.sharif.edu/courses/ssc/unix/resources/root/Slides/unixhistory.pdf
- http://www.cs.uga.edu/~eileen/1730/Notes/intro-UNIX.ppt
- http://remus.rutgers.edu/cs416/F01
- http://www.cs.virginia.edu/~cs458/
- http://www.bobbooth.staff.shef.ac.uk/hpcs/materials/material.html
- http://www.comm.utoronto.ca/~jorg/teaching/ece461
- http://home.iitk.ac.in/~navi/sidbilinuxcourse/
- http://www.cs.washington.edu/homes/bershad/Mac/ssh/practicalmagic.pdf
- http://www.cs.cf.ac.uk/Dave/C/CE.html
- http://www.le.ac.uk/cc/tutorials/c/ccccintr.html
- http://www.shef.ac.uk/uni/academic/N-Q/phys/teaching/phy225/index.html