ПОДГОТОВКА ЗА ТЕОРЕТИЧЕН ИЗПИТ ПО ОПЕРАЦИОННИ СИСТЕМИ, СИ, 2020/2021

Ден 1. (30.06.2021 г.)

1. ***Опишете с по едно-две изречения работата на следните системни извиквания в стандарта POSIX:***
   - pipe() – Създава неименувана тръба. Приема масив от две цели числа int fd[2], в който се съхраняват двата нови файлови дескриптора, отнасящи се до краищата на тръбата. fd[0] съхранява файловия дескриптор за четене, а fd[1] – за писане.
   - dup2() – Създава копие на файлов дескриптор, подаден като първи аргумент и го записва на номера на файлов дескриптор, подаден като втори аргумент. Ако вторият файлов дескриптор е отворен, той първо бива затворен преди да бъде използван повторно.
   - fork() – Създава нов процес, копирайки извикващия процес. Не приема аргументи и връща идентификатор на процеса – новият процес (детето) има стойност 0, а извикващия процес (бащата) има стойност по-голяма от 0. При грешка, връща стойност -1.
   - exec() – Системните извиквания от това семейство заменят изпълнимия файл на процеса, който се изпълнява в момента, с друг изпълним файл, подаден като първи аргумент. Следващите подадени аргументи могат да се интерпретират като масив от аргументи, които желаната команда да приеме за своето изпълнение.
   - wait() – Спира изпълнението на извикващия процес, докато някой от процесите-дете не прекрати своето изпълнение. При подаден аргумент, съхранява подробна информация за детето, променило своето състояние.
   - waitpid() – Спира изпълнението на извикващия процес, докато процесът-дете с идентификатор, равен на подадения аргумент, не промени състоянието си.
   - open() – Отваря (или създава) файл с име, подадено като аргумент. Връща като резултат стойността на файловия дескриптор, асоцииран с този файл, а при грешка връща -1.
   - close() – Затваря файлов дескриптор, асоцииран с даден файл.
   - read() – Приема три аргумента – файлов дескриптор, буфер и брой байтове. Опитва се да прочете съответния брой байтове от файловия дескриптор и да ги запише в буфера.
   - write() – Приема три аргумента – файлов дескриптор, буфер и брой байтове. Опитва се да прочете съответния брой байтове от буфера и да ги запише във файловия дескриптор.
   - lseek() – Приема три аргумента – файлов дескриптор, отместване и отправна точка. Премества „курсор“-а във файловия дескриптор, на позиция <отместване> според отправната точка. Има три вида отправна точка – SEEK_SET – от началото, SEEK_END – от края, SEEK_CUR – от текущото положение.
   - socket() – Използва се за създаване на комуникационен канал от тип „конекция“. Създава сокет (крайна точка за комуникация) и връща файловия дескриптор, асоцииран с него.
   - bind() – Присвоява адрес на сокет, създаден със socket(). Това извикване се използва за сървърите в конекцията, изградена със socket(), а при клиентите не е необходимо. Тази операция се нарича “именуване на сокет”.
   - connect() – Свързва сокета, посочен от файловия дескриптор, подаден като аргумент, към адреса на сокета, подаден като аргумент.
   - listen() – Означава сокета, подаден като аргумент, като готов да приема нови входящи заявки за свързване.
   - accept() – Блокиращо повикване, което чака входящи връзки. След като заявката за връзка бъде обработена, нов файлов дескриптор се връща. Този нов сокет е свързан с клиента (установена е конекцията), а другият сокет остава в състояние listen в очакване на допълнителни връзки.

2. ***Каква е спецификата на файловете в следните директории в Linux:***
   - /etc – съдържа конфигурационните файлове; файлове, които управляват конфигурацията на конкретната изчислителна система след зареждането на операционната система; информация за потребителите, пароли и т.н.
   - /dev – съдържа драйверите на системата; в нея се описват устройствата, които са част от изчислителната система; съдържа псевдо файлове, които задават какъв хардуер може да бъде обслужван от нашата система и какъв точно се обслужва
   - /var - системни файлове - logging files, mail directories, printer spool, etc.; за данни, не на потребителя, а за споделени данни
   - /boot – информация за ОС преди самото й стартиране; неща, свързани със зареждането на операционната система
   - /usr/bin – съдържа повечето изпълними файлове, които не са необходими за стартиране или поправяне на системата
   - /home – home директориите на потребителите в системата; файловете на конкретните потребители
   - /usr/lib – обектни файлове и библиотеки; може да включва и вътрешни двоични файлове, които не са предназначени да бъдат изпълнявани директно от потребители или shell скриптове
   - /var/log – директорията, съдържаща всички log (журнални) файлове
   - /proc – информация за операционната система в реално време; съдържа псевдо файлове със статистическа информация от ядрото
   - /bin – съдържа изпълними файлове; важни програми, които се изпълняват при стартирането на системата, общодостъпни програми
   - /usr/doc – съдържа всичката необходима документация.

3. ***Опишете функционалността на следните команди в Linux:***
   - cd – променя текущата директория
   - mkdir – създава директория
   - rmdir – премахва директория
   - cp – копира файл/файлове в посочена дестинация
   - mv – премества/преименува файлове в зададена дестинация
   - rm – премахва файлове
   - ls – показва съдържанието на директория
   - who – показва активните потребители в момента на извикване
   - find – търси файлове/директории по зададени параметри
   - ps – показва подробна извадка за процесите в ОС в момента на извикване
   - top – подобна на командата ps, показва подробна информация за процесите в ОС в реално време
   - vi – отваря на конзолата текстов редактор
   - tar – създава компресирани или некомпресирани архивни файлове или ги поддържа и модифицира
   - gcc – компилира програми, написани на C и C++ код
   - echo – показва на дисплея ред от текст/низ, подаден като аргумент/на стандартния вход
   - read – прочита ред текст, подаден на стандартния вход
   - test – проверява типове на файлове и сравнява стойности (числови или низове)
   - if – (if <условие>; then <списък команди> fi); изпълнява команди въз основа на подадени условия; ако подаденото <условие> е изпълнено, се изпълнява <списък команди>; тялото

на конструкцията приключва с думата fi; в случай че първоначалното условие не е изпълнено, позволява проверка на допълнителни условия с ключовата дума elif – от else if; ако никое от подадените условия не е вярно, в else могат да се запишат команди, които да бъдат изпълнени.

- for – (for <елемент> in <множество>; do <команди> done); изпълнява итеративно (циклично) набор от <команди> за всеки <елемент>, присъстващ в подадено <множество>
- while – (while <условие>; do <команди> done); изпълнява итеративно (циклично) набор от <команди> докато дадено <условие> е изпълнено
- chmod – променя правата на подадените файлове според подадените стойности (могат да бъдат представени както числово, така и символно)
- cat – прочита съдържанието на файл и го извежда на стандартния изход
- grep – търси редове, отговарящи на подаден образец (pattern), и извежда само съвпадащите редове на стандартния изход
- cut – извежда само избрани секции/части от редовете на подаден файл/текст; разделя редовете на отделни колони спрямо подаден символ за разделител (по подразбиране е спейс/табулация)
- sort – сортира редовете на подаден файл/текст и извежда сортираното съдържание на стандартния изход
- wc – извежда броя на новите редове, думите и байтовете за подаден файл
- tr – заменя символи в даден файл според подадено множество със съответните символи на същата позиция във второто множество или изтрива всички срещания на символите от първото множество

4. ***Семафор – дефиниция и реализация. Опишете разликата между слаб и силен семафор.***

- Семафорът е абстрактен механизъм от високо ниво за синхронизация на процеси, използващи общи ресурси. Основава се на принципа за приспиване и събуждане на процес. Ако един процес няма работа, която да извърши в даден момент, и очаква външно събитие, то той бива приспан до настъпване на чаканото събитие. Когато споделеният ресурс е наличен отново, процесът се събужда и продължава изпълнението си от там, докъдето е стигнал.
- Имплементация на семафор:

```
struct Semaphore {

        int cnt = 0
        list L = Ø
        bit lock = 0

        init(int _cnt) {
                cnt = _cnt
        }

        wait( ) {
                spin_lock(lock)
                cnt = cnt − 1
                if cnt < 0 {
                        L.put(self)
                        spin_unlock(lock)
                        block()
                } else spin_unlock(lock)
        }

        singal( ) {
                spin_lock(lock)
                cnt = cnt + 1
                if cnt ≤ 0 {
                        pid = L.get()
                        spin_unlock(lock)
                        wakeup(pid)
                } else spin_unlock(lock)
        }
}
```

- Структурите от данни, необходими за реализация на семафор са: Брояч cnt, в който се пази броя на процесите, които могат да бъдат допуснати до ресурса, охраняван от семафора. Контейнер Q, в който се пази информация кои процеси чакат достъп до ресурса.
- Процедурите (методите), необходими за реализация на семафор са: Конструктор init(int n), който задава начална стойност на брояча n и инициализира контейнера празен. Метод wait(), който се ползва при опит за достъп до ресурса. Броячът се намалява с единица и ако той стане отрицателен, процесът, извикал wait() се блокира, а номерът му се вкарва в контейнера Q. Метод signal(), който се използва при освобождаване на ресурса. Броячът се увеличава с единица и ако контейнерът Q не е празен, един от процесите в него се изкарва и активира.

- Един семафор е силен, когато контейнерът Q е реализиран като обикновена опашка (на принципа „first in, first out“ – активираме процеса, блокиран най-рано). Семафорът е слаб, когато контейнерът Q е реализиран като приоритетна опашка – активираме процес, който не е влязъл най-рано в него.

5. ***<u>Опишете условието на „Задача за читателите и писателите“ и решение, използващо семафори.</u>***
   - Условието на Задачата за читателите и писателите се изразява в следното: Имаме стая, в която читатели могат да влизат да четат, а писатели да пишат. Искаме да синхронизираме операциите за четене и писане по следния начин:
     - Ако в стаята има писател, то никой друг не трябва да има достъп до стаята.
     - В стаята може да има произволен брой читатели.
     - Не трябва да има условие за deadlock.
     - Не трябва да има условие за starvation на някой от писателите/читателите.
   - Идеята на решението на задачата е следната: Когато първият читател се опита да влезе в стаята (под стая разбираме критична секция), той трябва да провери дали е празна, след което всеки следващ читател е необходимо да проверява само дали в стаята има поне един читател. Възможно е обаче писателите да гладуват, заради наличието на много на брой читатели или бавни такива. Затова използваме бариера, която ще спира читатели да навлизат, при условие, че има поне един писател, който чака да влезе в критичната секция.

barrier.init(1);
mutex.init(1);
roomEmpty.init(1);
cnt=0	

Инструкции на READER:	Инструкции на WRITER:

barrier.wait()
barrier.signal()
mutex.wait()
cnt=cnt+1
if (cnt==1) roomEmpty.wait()
mutex.signal()
READ
mutex.wait()
cnt=cnt+1
if (cnt==0) roomEmpty.signal()
mutex.signal()
	barrier.wait()
	roomEmpty.wait()
	
	WRITE
	roomEmpty().signal()
	barrier.signal()

6. ***Опишете как се изгражда комуникационен канал (connection) между процес-сървър и процес-клиент със следните системни извиквания в стандарта POSIX: socket(), bind(), connect(), listen(), accept()***
   - Един процес, наричан обичайно сървър, изпълнява следната поредица:
     - sfd=socket(domain, type, protocol); // създава socket
     - bind(sfd, &my_addr, addrlen); // присвоява име на socketa
     - listen(sfd, backlog); // започва приемане на заявки за връзки
     - cfd = accept(sfd, &peer_addr, addrlen); // приема заяка за изграждане на връзка
   - Друг процес, наричан обичайно клиент, изпълнява следната поредица:
     - fd=socket(domain, type, protocol); // създава socket
     - connect(fd, &server_addr, addrlen); // подава заяка за изграждане на връзка
   - Сървърът създава сокета sfd и му дава име чрез bind. Извикването listen активира процеса на изграждане на връзки. Клиентът създава сокета fd, без да е нужно да го именува. Извикването connect е заявка за изграждане на връзка към именувания сокет sfd. Сървърът приема заявката на клиента чрез accept. Изградената връзка е между файловите дескриптори cfd на сървъра и fd на клиента. Те ги ползват за обмен на информация. Файловият дескриптор sfd на сървъра продължава да приема нови заявки от клиенти. Благодарение на присвоеното му име sfd дава възможност на другите процеси да се свържат със сървъра. Името на sfd определя какви клиенти могат да ползват сървъра. Ако то е име в интернет (IP адрес, порт), всички процеси, изпълнявани на компютри, имащи достъп до интернет, могат да се свържат към сървъра.

7. ***Опишете накратко основните комуникационни канали в ОС Linux. Кои канали използват пространството на имената и кои не го правят?***
   - Неименуваната тръба (pipe) се създава чрез системното извикване pipe(int fd[2]). То запазва в подадения масив два файлови дескриптора – fd[0] съхранява файловия дескриптор за четене, а fd[1] – този за писане. Тази тръба е видима само за процеса, който я е създал, и за неговите наследници, тъй като не е именувана – *не използва пространството на имената*.
   - Именуваната тръба (FIFO) се създава чрез извикването mkfifo(…). За разлика от неименуваната тръба, този вид тръба *използва пространството на имената* и е видима за всички процеси в системата и може да бъде ползвана от тях – може да бъде използвана от процеси, които нямат роднинска връзка.
   - Връзката процес-файл. Чрез системното извикване open(…) се създава файлов дескриптор, сочещ към края на комуникационния канал, който се изгражда в ядрото и отговаря за писане, а другият край на канала сочи към файла. Чрез извикванията read(..) и write(…) може да се чете и пише от/в този файл. Този вид канал също използва *пространството на имената*.
   - Конекцията (изградена с механизма socket) е именуван обект, който се използва за адрес при изграждането на връзка с друг процес. Тук можем да разделим процесите на два вида – сървър (този, който обслужва останалите) и клиент. При конекцията не е необходимо процесите да са на една и съща система. Сокетът на сървъра *използва пространството на имената*, тъй като клиентите трябва да го виждат по някакъв начин, но клиентите не се обвързват с име – *не ползват пространството на имената*.

Ден 2. (01.07.2021 г.)

8. ***Опишете разликата между синхронни и асинхронни входно-изходни операции. Дайте примери за програми, при които се налага използването на асинхронен вход-изход.***
   - При синхронна входно-изходна операция системното извикване може да доведе до приспиване (блокиране) на потребителския процес, поръчал операцията. Същевременно, при нормално завършване, потребителският процес разчита на коректно комплектоване на операцията – четене/запис на всички предоставени/поръчани данни във/от входно-изходния канал, или цялостно изпълнение на друг вид операция (примерно, изграждане на TCP връзка).
   - При асинхронна входно-изходна операция системното извикване не приспива (не блокира) потребителския процес, поръчал операцията. Същевременно, при невъзможност да се окомплектова операцията, ядрото връща управлението на процеса със специфичен код на грешка и друга информация, която служи за определяне на степента на завършеност на операцията. Потребителският процес трябва да анализира ситуацията и при нужда да направи ново системно повикване по-късно, с цел да довърши операцията. Използването на асинхронни операции позволява на един процес да извършва паралелна комуникация по няколко канала с различни устройства или процеси, без да бъде блокиран в случай на липса на входни данни, препълване на буфер за изходни данни или друга ситуация, водеща до блокиране.
   - Примери за асинхронни входно-изходни операции: Когато ползваме WEB-browser, той трябва да реагира на входни данни от клавиатура и мишка, както и на данните, постъпващи от интернет, т.е. на поне 3 входни канала. Браузерът проверява чрез асинхронни опити за четене по кой от каналите постъпва информация и реагира адекватно. Сървър в интернет може да обслужва много на брой клиентски програми, като поддържа отворени TCP връзки към всяка от тях. За да обслужва паралелно клиентите, сървърът трябва да ползва асинхронни операции, за да следи по кои връзки протича информация и кои са пасивни.

9. ***Дайте кратко определение за: многозадачна ОС, многопотребителска ОС, времеделене.***
   - *Многозадачната ОС* изпълнява няколко задачи едновременно. Това се постига чрез разделение на времето на работа на процесора според инструкциите на специална подсистема (task scheduler). При разпределена многозадачност, ОС отпуска на задачата определено време да ползва процесора. Ако тя не успее да приключи за това време, ОС я форсира да отстъпи процесора на следващата задача, която се нуждае от него. При кооперативната многозадачност, приложението, стартирано от ОС, използва 100% от процесора. В този случай, ако друга програма изиска процесорно време, то или няма да ѝ бъде предоставено, което ще доведе до нарушаване на функциите на това приложение и/или до терминирането му, или ще бъде предоставено след приключване на първото.
   - *Многопотребителската ОС* разширява концепцията за многозадачност, като различават потребителите по отношение ползването на процеси и ресурси, като например дисково пространство. Те планират ефикасното използване на ресурсите на системата и могат да съдържат специализиран софтуер за изчисление на процесорното време от много потребители, както и да отчитат използваната памет, ползване на принтер и други използвани ресурси.
   - *Времеделенето* позволява създаването на многопотребителски системи, в които централният процесор и блокът на оперативната памет обслужват много потребители. При това част от задачите (като въвеждане или редактиране на данни) могат да се

изпълняват в диалогов режим чрез терминали, а други (като обемните изчисления) – в пакетен режим.

10. <u>Опишете ситуацията съревнование за ресурси (race condition), дайте пример.</u>

- Race condition, буквално може да се преведе "борба за ресурси" е състояние, което настъпва когато два или повече процеса едновременно се опитат да достъпят даден споделен ресурс, като започва надпревара за това кой процес ще достъпи паметта първи. Това може да доведе до проблем, тъй като не само, че не знаем кой процес ще изпълни съответната "критична част" първи, ами и ако например споделения ресурс е дадена променлива и единия от процесите изпълнява следния код: if (cnt==0) x=1; Възможно е между проверката и присвояването на новата стойност друг процес да достъпи споделения ресурс, при което е възможно да настъпи проблем и в двата процеса.

11. <u>Опишете накратко инструментите за избягване на race condition:</u>

- дефинирайте критична секция, атомарна обработка на ресурса.
    - Критична секция: Всеки процес, който работи със споделен ресурс има поне едно място в кода си, в което той използва този споделен ресурс. Тази част от кода се нарича критична секция.
    - Атомарна обработка на ресурса: Повече от една инструкции, които променят бит и го прочитат след това като една единна непрекъсната инструкция.
- инструменти от ниско ниво, специфични хардуерни средства.
    - Най-простият инструмент за избягване на race condition е spinlock. Това е инструмент на най-ниско ниво, който на практика ни дава възможност да "заключим" достъпа до даден ресурс, когато го използваме ние, и да го "отключим" в последствие, когато вече не го използваме. Това става с наличието на един допълнителен бит, който ни указва дали ресурса е свободен за използване, или не е. Това обаче не е достатъчно, тъй като ако даден процес се извика рекурсивно, то всеки следващ процес ще зацикли и бита никога няма да се освободи, т.е. паметта ще остане винаги заключена, т.нар. deadlock. Освен този бит, е важно да се забранят прекъсванията, така че да не може да се стигне до зацикляне. В spinlock присъстват следните хардуерни инструменти за защита – enable/disable interrupt, test-and-set и atomic swap.
- инструменти от високо ниво, които блокират и събуждат процес.
    - Семафорът е абстрактен механизъм от високо ниво за синхронизация на процеси, използващи общи ресурси. Основава се на принципа за приспиване и събуждане на процес. Ако един процес няма работа, която да извърши в даден момент, и очаква външно събитие, то той бива приспан до настъпване на чаканото събитие. Когато споделеният ресурс е наличен отново, процесът се събужда и продължава изпълнението си от там, докъдето е стигнал.

12. <u>Хардуерни инструменти за защита (lock) на ресурс:</u>

- enable/disable interrupt: Инструкцията блокира временно прекъсванията. Когато се влезе в режим на работата в ядрото, кода извиква процедура от ядрото да го изпълни и може да се сложи в опашката инструкция да спре и да пусне прекъсванията.
- test and set: Тази инструкция се използва, за да се пише в паметта и да се върне старата стойност като атомарна операция.
- atomic swap: Атомарната инструкция се използва да се постигне синхронизация. Тя сравнява съдържанието на локация в паметта с дадена стойност и само ако са същите, модифицира съдържанието с новата стойност.

13. <u>Опишете предназначението на журнала на файловата система.</u>

- Записваме промените във файл, който се нарича журнал – в него се запазва пълната информация за операциите над файлове и когато се запълни журналът, спираме и отразяваме операциите от журнала над файловете. Ако спре токът, журналът ще пази последните промени, а файловете ще са консистентни. Четат се първо старите файлове и се проверява дали в журнала са променени. Ако спре токът, докато пишем операцията/транзакцията, тя няма да се е изтрила преди да е завършила и затова ще я пазим все още в журнала. Във файловите системи транзакцията е елементарна файлова операция (read, write, open). Журналът (log файл) се намира или в друг диск или в друг дисков дял, за да може двете паралелно да работят, но в персоналните устройства журналът е файл в самата файлова система или в същия дял.

14. <u>Опишете накратко различните видове специални файлове в Linux:</u>

- *външни устройства, именувани в /dev* – файловете в тази директория са псевдофайлове, съответстващи на периферни устройства; задават какъв хардуер може да бъде обслужван от нашата система и какъв точно се обслужва;
- *псевдофайлове в /proc* – В директорията /proc е вградена виртуалната файлова система Proc file system (procfs), която се създава в движение при стартиране на системата и се разтваря в момент на спиране на системата. Тук се съдържа полезна информация за процесите, които се изпълняват в момента – разглежда се като контролен и информационен център за ядрото. В директорията има директории с име PID на всеки процес в системата, всяка от които съдържа подробна информация за този процес.
- *линкове* – твърди и символни, команда ln
    - Symbolic link – псевдофайл, който представлява алтернативно име за друг файл или директория. Те могат да сочат към всички видове файлове (нормални файлове, директории, специални файлове, други символни линкове и т.н.). Ако бъде преименуван, преместен или изтрит оригиналният файл, линкът става невалиден (счупен; broken). Те могат да имат различни права на достъп от файловете, към които сочат.
    - Hardlink – За разлика от symbolic links, които сочат към името на друг файл, твърдите линкове се насочват към inode-а на съществуващ файл. По този начин различни имена на файлове се свързват към един и същ inode. Не могат да сочат към директории. Преименуването, преместването или изтриването на „оригиналния“ файл няма ефект върху тях. Промяната на правата на достъп на файл променя правата на достъп на всички негови твърди линкове.
    - ln – създава линкове към файлове; ако не подадем на командата никакви флагове, тя създава hardlink на подадения файл; ако подадем на командата -s, тя създава symbolic link на подадения файл
- *сокет* - специален файл, използван за комуникация между процесите (инструмент за създаване на комуникационен канал от тип конекция), който позволява комуникация между два процеса без роднинска връзка. За разлика от именуваните тръби, които позволяват само еднопосочен поток от данни, сокетите са напълно съвместими с дуплекс. При изпълнение на команда ls -l, сокетите са отбелязани със символ ‚s‘ като първа буква на символната репрезентация на правата за достъп.

15. ***Опишете какви атрибути имат файловете в съвременна файлова система, реализирана върху блочно устройство (block device).***
    - Освен име и съдържание, които са напълно задължителни атрибути, всеки файл е придружен и от други атрибути (незадължителни или задължителни) – права за достъп, информация за собственост, група, размер, дата на създаване, дата на последна промяна, дата на достъпване.

16. ***Права и роли в UNIX***
    - Права – u/g/o user/group/others
        - Всеки файл и директория имат 3 вида групи по отношение на правата: Потребител (User/Owner), Група (Group) и Други (others).
            - Потребителят е собственикът на файла. По подразбиране, при създаване на файл, лицето, създало файла, става негов собственик.
            - Групата съдържа множество потребители. Всички потребители, принадлежащи към групата, на която принадлежи файлът, имат едни и същи права за достъп.
            - Други са всички останали потребители - нито са собственици на файла, нито принадлежат към потребителската група, притежаваща файла. На практика това означава всички останали.
    - Роли – r/w/x read/write/execute
        - Всеки файл и директория имат 3 вида роли: Четене (Read), Писане (Write) и Изпълнение (Execute).
        - Четене: Дава правото да отваряме и да четем съдържание на даден файл. При директориите, дава възможност да видим нейното съдържание (например с командата ls).
        - Писане: Дава право да променяме съдържанието на файл. При директориите, дава право да добавяме, премахваме и преименуваме файлове, съхранявани в директорията (ако съответните файлове също го разрешат).
        - Изпълнение: Ако разрешението за изпълнение не е зададено, не можем да стартираме изпълним файл. При директориите ситуацията е малко по-особена. Ако сравним с правата за четене, там можем само да видим съдържанието на дадената директория (например с команда ls), но ако нямаме права за изпълнение за дадената директория, не можем да я достъпваме.
    - chmod – правата за различните роли и групи се сменят с командата chmod. Тя приема като аргументи символна или числена репрезентация на новите права, които искаме да зададем за конкретния файл, както и самия файл, който искаме да променим.

17. ***Взаимно блокиране (deadlock)***
    - Deadlock е събитие, което се настъпва, когато процес чака друго събитие, което никога няма да се случи. Обикновено това се получава, когато нишка/процес чака мютекс или друг семафор, който никога не се освобождава от предишния си ползвател, както и при ситуации, когато имаме два процеса и две заключвания.

Thread 1:	Thread 2:
Lock1 -> lock() WaitForLock2()	Lock2 -> lock() WaitForLock1()

    - Има четири условия, които трябва да бъдат изпълнени, за да можем да сме сигурни, че ще възникне deadlock:

- Условие за взаимно изключване. Поне един процес да използва споделен ресурс монополно, т.е. да има ексклузивни права над него (да няма възможност друг процес да наруши работата му).
- Условие за задържане и изчакване. Може да се случат поредици при които процес взима даден ресурс и чака да вземе следващия.
- Ползването на споделен ресурс в първото условие да не може да бъде прекъсвано от външен фактор.
- Условие за кръгово (циклично) изчакване.

18. ***Алгоритъм на асансьора***

- Алгоритъмът на асансьора се изразява в следното: Разместват се така заявките към четене и писане, че главата да изминава минимално разстояние; Има две приоритетни опашки: едната се състои от файлове, които се намират по посока на движението на главата, а другата от файлове, които се намират в обратната посока; Ако е празна главата на опашката по посока на главата на устройството, то се сменя посоката.

19. ***Ефективна обработка на адресацията – MMU, TLB.***

- MMU (Memory management unit) – е хардуер, през който всички референции на паметта преминават, като главно извършват транслацията от виртуален адрес в физически адрес. Модерните MMU разделят виртуалното адресно пространство в страници, всяка от които има размер, който е степен на 2, обикновено няколко килобайта, но може и да са по-големи. Най-долните битове на адреса остават непроменени. Горните битове на адреса са числата на виртуалната страница.
- TLB (Translation lookaside buffer) – кеш, който хардуера за управление на паметта използва, за да подобри скоростта при транслация на виртуални адреси. TLB има фиксиран брой слотове, съдържащи записи от таблицата със страници и от таблицата със сегменти; Записите от таблицата със страници се използват за преобразуване на виртуалните адреси във физически адреси, а записите от таблицата със сегменти – за преобразуване на виртуалните адреси в сегментни адреси.

20. ***Превключване в система с времеделене – timer interrupt.***

- Работещ (Running) процес може да промени състоянието си на блокиран (да му бъде отнето процесорното време), ако твърде много време прекара в процесора. В такъв случай времето му изтича и той бива прекъснат от таймер, за да освободи CPU ресурс. Това е така заради времеделенето – всеки процес има максимално количество време, което може да работи. Обслужването на спирането се извършва от алгоритъм в ядрото, а самото спиране – от часовника. Активният (Ready) процес, преминава в работещ (Running), когато му се предостави процесорно време. Извършва се смяна на работещия процес, поради изтичане на време на даден процес. Ядрото тогава решава, кой чакащ процес да заработи (зависи от алгоритъма на ядрото и от неговия Task Scheduler). Когато спящият процес е приспан заради очакване на момент от времето, то той може да стане активен, като този процес се инициира от timer (прекъсване на часовника).

21. ***<u>Опишете как ОС разделят ресурсите на изчислителната система, дайте примери за основните типове разделяне: Разделяне на пространството (памети) и Разделяне на времето (процесори, други у-ва).</u>***

- При разпределянето на процесорното време голяма роля играе типът на процеса.
    - Има няколко вида процеси:
        - foreground/IO процеси – това са процеси, които извършват много входно-изходни операции (например чакат потребителя да въведе данни, да натисне бутон и т.н.). Тези процеси не се нуждаят от много процесорно време, но за тях е характерно, че често ще бъдат приспивани, т.е. ще прекарват много време в структурите между Running и Active и за тях е важно, когато бъдат събудени, много бързо да преминават към Running състояние, т.е. много бързо да им се дава процесорно време, когато го поискат.
        - background/CPU процеси – това са процеси, които извършват много изчисления, т.е. те няма често да бъдат приспивани. Характерно за тях е, че имат нужда от много процесорно време, но няма такова значение кога ще им бъде предоставено, т.е. няма значение кога ще им се даде възможност да преминат към Running състояние, но е важно, когато са в Running състояние, те да прекарат достатъчно време там.
        - real-time процеси - за тях е характерно, че имат определен срок (deadline), в който трябва да им се предостави процесорно време; такива процеси често се срещат например в медицинските роботи, където трябва да се извърши дадена операция и редът и времето на изпълнение на процесите са от критична важност.
    - Важно е предоставянето на процесорно време да бъде съобразено с типа на процесите. В днешните системи за определянето на реда, по който се предоставя процесорно време на процесите се извършва от т.нар. диспечер или Task scheduler. Най-простият начин е процесите, които искат процесорно време, да се пазят в обикновена опашка, като първият процес, който е пожелал процесорно време, е и първият, на който се предоставя. Сигурно е, че няма да настъпи гладуване, т.е. на всеки процес след определен квант време ще бъде предоставен процесор. Проблемът е, че не се взима под внимание типът на процесите. Ако даден foreground (I/O) процес поиска процесорно време, той трябва да изчака всички процеси преди него да завършат, за да получи възможност за работа. В днешните системи се използват много по-сложни начини за определяне на реда. Подход за справяне с тази ситуация е употребата на приоритети на процесите – на всеки процес присвояваме приоритет 1000 и при всяко изпълнение на процеса, от този приоритет се изважда времето, за което е работил, и след него процесорно време се дава на процеса с най-висок приоритет. Така, ако даден background (CPU) процес е чакал прекалено дълго, със сигурност ще му бъде дадено процесорно време в определен момент. Друг вариант са fixed и floating приоритетите. Всеки път, когато на процесор не му бъде дадено процесорно време, се увеличава неговият floating приоритет и при избор на следващ процес се взима процесът с най-висок приоритет.
- При разделяне на пространството от значение са не само активните процеси, но и приспаните, тъй като те все пак се нуждаят от даден блок от памет, където да съхраняват данните които използват. Ресурсът се разделя на части и всяка програма или потребител получава част от него. Проблемите, които ОС трябва да решава са следните: Да следи

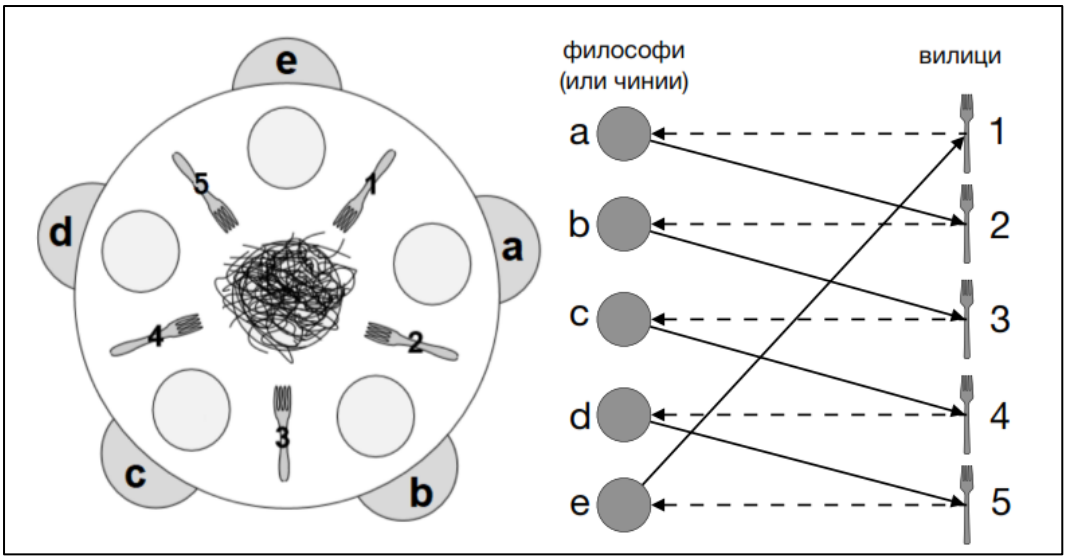
свободните и заети части, да осигури защита на частите, справедливо да разделя ресурса. Примери за ресурси, управлявани по този начин, са оперативна памет и дискова памет. Има два вида разделяне на паметта - статично и динамично; при статичното трябва да се знае каква е максималната големина на всеки файл, като се заделят n последователни блока, в които може да се запише файлът. Това осигурява много бърз достъп до данните, но на практика при повечето файлове не знаем каква ще бъде големината им и искаме да можем да се увеличават, динамично; при втория тип, файлът се разделя на т.нар. фрагменти с определена дължина, като тези фрагменти се разхвърлят по паметта (процесът се нарича фрагментация); по този начин не се пази максимална големина на файла, и нарастването му няма да доведе до проблеми.

22. <u>Отлагане на записа</u>

- Записваме промените във файл, който се нарича журнал. В журнала се запазва пълна информация за операциите над файлове и при запълване, спираме и отразяваме операциите от журнала над файловете. Ако спре токът, журналът ще пази последните промени, а файловете ще са консистентни (т.е. няма да настъпят повреди). Четат се първо старите файлове и се проверява дали в журнала са променени. Ако спре токът, докато пишем операцията/транзакцията, тя няма да се е изтрила преди да е завършила и затова ще я пазим все още в журнала. Във файловите системи транзакцията е елементарна файлова операция (read, write, open). Журналът (log файл) се намира или в друг диск, или в друг дисков дял, за да може двете паралелно да работят, но в персоналните устройства журналът е файл в самата файлова система или в същия дял.

23. <u>Задача за философите и спагетите</u>

- Разглеждаме кръгла маса, на която са седнали 5 философа и се опитват да се нахранят. По средата на масата има купа със спагети, а пред всеки философ има по една чиния и между всеки две чинии има вилица. За да си сложи от спагетите в чинията и да се нахрани, на даден философ са му необходими две вилици едновременно.
- Правим допускането, че философите знаят как да мислят и как да се хранят. Нашата работа е да напишем версия на разпределение на ресурсите (в нашия случай вилиците), в която да са изпълнени следните синхронизационни ограничения:
    - Когато един философ е взел вилица, никой друг не може да я ползва;
    - Да не е възможно да се появи deadlock;
    - Да е възможно повече от един (в нашия случай – повече от двама) философи да се хранят по едно и също време;
    - Да не е възможно философ да гладува дълго време, докато чака за вилица.
- Ако всеки философ се опита да вземе първо лявата или дясната вилица, ще настъпи deadlock, защото, за да се нахрани са му нужни две вилици, които да използва едновременно и всеки философ ще чака и ще гладува вечно.
- Можем да моделираме лесно тази ситуация с двуделен граф. Процесите ще са философите, а ресурсите ще са вилиците. С ребрата в графа ще обозначаваме необходимостта от ресурс на процес, за да може да си свърши работата.



- Нека разгледаме горепосочената ситуация, в която всеки философ си е взел лявата вилица и се опитва да вземе дясната. Ще насочим ребрата на графа, за да обозначим това, като посоката ще е от процес към ресурс, а пък обратната посока (пунктираната част) ще означава, че процесът се опитва да вземе този ресурс, но той е зает. При такова ориентиране на ребрата се получава цикъл и това води до deadlock.
- Нека номерираме множеството от процеси и множеството от ресурси, така че да знаем на кой процес кои ресурси са необходими. След като знаем на кой процес какво множество от ресурси е необходимо, за да си свърши работата, то той ще може да си го сортира и да ги взема по реда на номерата. Тогава алгоритъмът няма да е: „всеки да взима лявата вилица“, а ще е „всеки да взима вилицата с по-малък номер“. Така deadlock няма да настъпи (естествено има малка вероятност за гладуване на някои философи, но deadlock няма да има).

24. <u>Опишете понятията приспиване и събуждане на процес (block/wakeup).</u>

- Когато даден процес чака определено събитие да настъпи, независимо дали това събитие е някакъв момент във времето, някакви входно-изходно операции, или конкретно събитие, той бива приспан (block). Обикновено процесът се приспива сам, като извика block(), а по-късно, когато настъпи конкретното събитие, някакъв друг процес/сигнал го събужда (wakeup).

25. <u>Опишете разликата между многопотребителска и многозадачна работа. Какви качества на ОС характеризират тези две понятия?</u>

- Разликата между многопотребителска и многозадачна операционна система е, че при многозадачната ОС имаме един потребител, който едновременно изпълнява няколко задачи. При многопотребителската система имаме множество от потребители, всеки от които изпълнява някакви задачи, като се предоставя възможност на всеки от потребителите да работи, като по този начин изглежда, че за всеки потребител има отделно ядро, което всъщност не е така.

26. <u>Опишете разликата между времеделене и многозадачност.</u>

- Времеделенето е споделяне на изчислителен ресурс между много потребители чрез едновременна многозадачност, докато многозадачността е едновременно изпълнение на множество задачи или процеси за определен период от време. Докато споделянето на времето позволява на множество потребители да използват компютърна система

едновременно, многозадачността позволява на множество задачи или процеси да използват компютърна система наведнъж.

27. ***Ефективна реализация на файлова система.***

- Твърд диск – това е диск разделен на много „пътечки“:
  - Над повърхнината има устройство, което засича дали битът е 1 или 0;
  - Придвижването на главата на това устройство от една пътечка на друга е механично и става бавно;
  - Всяка пътечка е разделена на няколко сектора от по 512 или 1024 байта;
  - Главата не се допира до диска, а лети много малко над повърхността;
  - Софтуера знае колко е голям сектора и колко е времето за преминаване от един до друг сектор;
  - Файла се представя като много сектори;
  - Не са последователно разпределени байтовете на файловете, защото така не могат лесно да нарастват;
  - Файлът се разполага там където може, процесът на разхвърляне се нарича фрагментация.

28. ***Какви ресурси разделя еднозадачна, еднопотребителска ОС?***

- Еднозадачната еднопотребителска ОС е операционна система, която позволява на един потребител да изпълнява само една задача наведнъж. Функции като отпечатване на документ, изтегляне на изображения и т.н., могат да се изпълняват само по една. Тази операционна система заема по-малко място в паметта. Еднозначната еднопотребителската заделя всички ресурси за този един потребител и тази една задача.

29. ***Гладуване (livelock, resource starvation).***

- *Livelock* се случва, когато два или повече процеса непрекъснато повтарят едно и също взаимодействие в отговор на промените в другите процеси, без да извършват полезна работа. Тези процеси не са в състояние на изчакване и те се изпълняват едновременно. Това е различно от deadlock, тъй като при deadlock всички процеси са в състояние на изчакване, а тук те са „живи“ (все едно вършат дадена работа), но всъщност не вършат нищо полезно.
- *Starvation* настъпва, когато „алчните“ (greedy) нишки използват споделените ресурси прекалено дълго и по този начин ги правят недостъпни за останалите за дълги периоди от време. Да предположим например, че обектът предоставя синхронизиран метод, който често отнема много време, за да се върне. Ако една нишка често извиква този метод, други нишки, които също се нуждаят от чест синхронизиран достъп до същия обект, често ще бъдат блокирани. Обикновено гладуването е причинено от прилагането на твърде опростен алгоритъм за съставяне на разписания, по които процесите се изпълняват – не се взима под внимание типът на различните процеси.

30. ***Единна йерархична файлова система в UNIX.***

- В UNIX абстрактната файлова система е кореново дърво. Коренът представя началото на файловата система. Върховете представляват директории, а листата - файлове, като всеки връх (директория) също представлява кореново дърво. Възможно е две файлови системи да бъдат закачени една към друга, като това се осъществява чрез командата mount, която монтира (закача) едната файлова система към другата. Командата unmount служи за откачане на една файлова система от друга. Има някои специални директории, които присъстват във всяка файлова система (напр. /dev, /etc, /tmp и т.н.) и служат за

правилната организация и работа на системата. Към всеки файл, освен името и съдържанието, се асоциират и други атрибути - тип на файла ("-" – обикновен файл, "d" – директория, "c" – символно устройство/character special device (напр. клавиатура), "b" – блоково устройство/block special device (външно устройство, което съхранява масив от байтове), "p" – именувана тръба (FIFO), "s" – socket/конектор (крайна точка за комуникация)), права за достъп, потребител-собственик, потребителска група, размер, дата на промяна, дата на създаване и др. В Linux тези атрибути се пазят отделно от самия файл, което от една страна прави достъпа до тях по-бавен, но от друга страна дава възможност за съществуването на т.нар. hardlinks, т.е. за създаване на друго име на файла (файлът ще бъде същия, понеже ще сочи към същите атрибути, но името му може да бъде различно). По този начин, файлът ще бъде изтрит от паметта чак когато всички hardlinks към него се изтрият.

31. ***<u>Комуникационна тръба (pipe) с буфер – тръба, съхраняваща n пакета информация. Използване на семафорите като броячи на свободни ресурси.</u>***

- Тръбата се ползва от няколко паралелно работещи изпращачи/получатели на байтове. Процесите изпращачи слагат байтове в края на опашката, получателите четат байтове от началото на опашката.
- Необходими структури от данни:
  - Опашка (или масив) Q с n елемента – тъй като е възможно да имаме бърз и бавен процес (например: единият чете бързо, а другият пише бавно), е нужно опашката да бъде по-дълга, за да не се приспива по-бързият процес (приспиването е бавна операция, тъй като включва в себе си смяна на контекста). В началото тази опашка е празна;
  - free_bytes – семафор, който ще индикира за свободните байтове в опашката;
  - ready_bytes – семафор, който ще индикира колко байта са готови за четене (до колко е запълнена опашката);
  - mutex_read – мутекс, който ще защитава критичната секция за четене;
  - mutex_write – мутекс, който ще защитава критичната секция за писане.
- Процесът, който чете, първоначално ще проверява дали има готови за четене байтове. Ако няма, той ще се приспи и ще чака да постъпят такива. Ако има, той ще провери дали някое друго копие на P не чете в момента. Ако да, той отново се приспива. Ако не, той ще прочете байт и ще го запише в променливата. След това ще сигнализира, че в опашката има още един свободен байт чрез подаване на сигнал на семафора free_bytes.
- Процесът, който пише, първоначално ще провери дали има свободни байтове в опашката. Ако няма, ще се приспи и ще чака да се освободят байтове. Ако има, ще трябва да провери дали някое друго копие на пишещ процес не пише в момента в опашката. Ако да, то отново процеса ще се приспи. Ако не, ще сложи байта си в опашката и ще сигнализира, че още един байт е готов за четене, което се осъществява чрез подаване на сигнал на семафора ready_bytes.

Инициализация: free_bytes.init(n) ready_bytes.init(0) mutex_read.init(1) mutex_write.init(1)	
<u>READ:</u>	**<u>WRITE:</u>**
byte b p1 p2 … ready_bytes.wait() mutex_read.wait() b = Q.get() mutex_read.signal() free_bytes.signal() …	byte b q1 q2 … free_bytes.wait() mutex_write.wait() Q.put(b) mutex_write.signal() ready_bytes.signal() …

32. Взаимно изключване – допускане само на един процес до общ ресурс. Опишете решение със семафори.

- При взаимното изключване (mutual exclusion или mutex) искаме най-много един процес да достъпва определен споделен ресурс по едно и също време. Например, нека p1, p2 и p3 формират критичната секция на процеса P и нека P има много копия, т.е. множество копия на P могат да работят паралелно. Искаме само едно копие на P да може да достъпва критичната секция по едно и също време.
- Тази синхронизационна задача може да се реши като инициализираме семафор s и добавим следните синхронизационни инструкции в кода:

```
s.init(1);
P
s.wait()
p1
p2
p3
s.signal()
```

- По този начин, първото копие на процеса P ще започне да изпълнява критичната си секция, като брояча на семафора ще стане 0. След това, всяко следващо копие, което се опита да достъпи критичната секция ще бъде приспивано, докато първото не приключи изпълнението си, и не събуди едно от приспаните копия. И така всеки следващ събуден процес след приключване на своята критична секция ще събужда следващ приспан процес и т.н.

33. Опишете инструмента spinlock, неговите предимства и недостатъци.

- Това е инструмент на най-ниско ниво, който на практика ни дава възможност да "заключим" достъпа до даден ресурс, когато го използваме ние, и да го "отключим" в последствие, когато вече не го използваме. Приемаме, че хардуерът разполага със специални инструменти за атомарна обработка на ресурсите, напр. test_and_set(), който атомарно (като една цяла инструкция) да проверява каква е текущата стойност и съответно да присвоява нова стойност. Базирайки се на това допускане, можем да дефинираме друг инструмент от ниско ниво за справяне с race condition - т.нар spinlock. При него имаме допълнителен бит – lock, който пази стойност 0 ако ресурсът е свободен, и стойност 1, ако е зает. Тъй като този бит е споделен ресурс, за обработката му ще използваме test-and-set. Реализацията на spinlock до този момент би изглеждала така:

```
spinlock R = test_and_set(lock)
if(R==1) goto spinlock
 critical_section
unlock(lock)
```

- Тук обаче е възможно да възникне проблем, ако процесорът прекъсне изпълнението на процеса, затова е важно да се забранят прекъсванията.

```
spinlock disable_interrupts
R=test_and_set(lock)
if(R==0) goto critical_section
enable_interrupts
goto spinlock
critical_section
unlock(lock)
enable interrupts
```

- Важно е да се отбележи, че не можем да имаме рекурсия в критичната секция, защото по този начин извикания процес ще чака lock-а да се освободи, а текущия процес ще чака рекурсивно извикания да приключи своето изпълнение - което никога няма да се случи, т.е. ще настъпи deadlock.

34. <u>Опишете какви изисквания удовлетворява съвременна файлова система, реализирана върху блоково устройство (block device).</u>

- Твърд диск – това е диск разделен на много „пътечки“
- Над повърхнината има устройство, което засича дали битът е 1 или 0;
- Придвижването на главата на това устройство от една пътечка на друга е механично и става бавно;
- Всяка пътечка е разделена на няколко сектора от по 512 или 1024 байта;
- Главата не се допира до диска, а лети много малко над повърхността;
- Софтуера знае колко е голям сектора и колко е времето за преминаване от един до друг сектор;
- Файла се представя като много сектори;
- Не са последователно разпределени байтовете на файловете, защото така не могат лесно да нарастват;
- Файлът се разполага там където може, процесът на разхвърляне се нарича фрагментация.

35. <u>Комуникационна тръба (pipe), която съхранява един пакет информация – реализация чрез редуване на изпращача/получателя.</u>

- Ако трябва да използваме семафори, за да реализираме комуникационна тръба (pipe), която съхранява само 1 пакет информация, решението би изглеждало така:

s1.init(1) s2.init(0)	
WRITER:	READER:
s1.wait() .. WRITE .. s2.signal()	s2.wait() .. READ .. s1.signal()

- Идеята на решението е, че тъй като може да се записва само по един пакет, трябва да се редуват изпълненията на 2-та типа процеси. Така процесът, който чете от тръбата трябва да изчака нещо да бъде записано вътре и се приспива, докато това не се случи. Първият процес записва в тръбата един пакет информация. Всеки следващ процес, който пише в тръбата ще бъде приспан, докато процес от другия тип не прочете записаното в тръбата. След като го прочете той ще сигнализира на процеса писател, като един от приспаните ще се събуди и ще запише нещо.

36. <u>Процеси в многозадачната система.</u>

- Процесът е работеща програма, съществуваща във времето, която има начало и край. Има различни състояния: R, A, S. Процесите могат да бъдат спящи (те чакат входно-изходни операции или момент от времето), активни (в момента активен процес, но който чакат CPU) и работещи(такива, които ползват CPU в момента).

37. <u>Превключване, управлявано от синхронизация.</u>

- Един процес преминава от едно състояние в друго:
  - Преход: Работещ процес да премине в спящо състояние. Този процес се нарича блокиране. Това настъпва, когато процесът чака входно/изходна операция (wait – предизвиква се от синхронизиращия механизъм - семафора, който от своя

страна приспива процеса, само ако ресурсът е зает) или ако чака момент от времето (sleep – предизвикан от синхронизираща операция, която задължително го приспива).
    - Спящият процес може да премине в активен (да се събуди). Събуждането на процеса се предизвиква от завършването на входно-изходна операция на друг процес, който чрез signal() ще подаде сигнал, че е освободен ресурс.

38. Процес и неговата локална памет – методи за изолация и защита.

- При управлението на паметта се използва статистика за използване на паметта. Хубавата ОС трябва да може динамично да разпределя наличната физическа памет между процесите и да се съобрази с тяхната активност и до колко те използват паметта, за да организира ефективно изчислителния процес.
- Механизми за защита на ползваната памет:
    - Разделяне на паметта на интервали – най-простият механизъм. Разглеждаме паметта като непрекъснат блок, но я разделяме на интервали като всеки процес има част. Процесите могат да ползват само дадения им блок памет. Когато трябва да комуникира с друг процес, той се обръща към ядрото. С хардуерни механизми се защитава паметта и се забранява ползването на други парчета памет. Най-простият механизъм е да има специални регистри, в които се указва къде е този интервал. В ранните машини интервалът се показва пряко (с начало и край). При всяко четене на памет трябва да се повери дали адресът на паметта, която достъпваме е в този интервал. Необходими са 2 сравнения, за да се провери дали се нарушават ограниченията. (S – start, E – end; S <= A < E)
    - Управлението на паметта предлага защита чрез използването на 2 регистъра - базов регистър и регистър с ограничение. Базовият регистър държи най-малкия легален физически адрес в паметта, а регистърът с ограничение указва размера на интервала. Това е така наречениият защитен интервал (base, size). Всеки адрес, който ползва потребителския процес, може да се разглежда в интервала (от 0 до Size – виртуален адрес). Реалният адрес a -> A = Base + a; a < size – проверка дали се излиза от интервала. Така се проверява дали сме в собствения блок от паметта или нарушаваме границите.
- В по-ранните системи и досега за реализиране на ефективна защита на паметта се използва предоставяне на интервал от паметта и новото е, че тя се разбива на страници (малък брой стандартни единици).
- Хардуерни инструменти: сегментация – програмата не се разглежда като 1 интервал, а като възможност за работа в няколко интервала, наричани сегменти; разбиване на таблици (paging) - в хардуера има таблица, която казва за всяка страница кой има права да я ползва.

39. Йерархия на паметите – кеш, RAM, swap.

- Паметта я разглеждаме като абстракция с реалността (паметта като единен блок, поредица от байтове). При бърза работа с паметта – скъпо в енергиен план; те са енергозависими. Бързите памети са също така и по-нетрайни. При междинният клас памети (динамични RAM чипове) – евтини, относително бързи и събират много информация, но ако не биват често обновявани, паметта се губи, защото се губи сигналът поради факта, че много малко електроника се влага в RAM.
- Кеш памет – спомагателна памет за ускоряване обмена на данни между различните нива в йерархията на паметта; скъпа, но бърза; подходяща за пресмятане. Ускоряването се постига чрез поддържане на копия от избрани части от данните върху носител с бързо действие, близко до това на горното ниво на паметта. За да са по-ефективни и по-ефикасни в употребата на данни, кешовете са сравнително малки.

- RAM памет – паметта, която процесора директно ползва; По-голямата част от процеса е тук. В нея се разполагат стекът, кодът на програмата, данните (така бързо става четенето и писането на байтове).
- Хард дискът – вечен, но те са механични устройства. Следователно са тежки, хабят енергия, подлежат на опасност от физически повреди.
- Swap – механизъм, чрез който процес може да бъде преместен временно от главната памет в резервен компонент (хард диск) и после да бъде върнат пак в паметта, за да продължи изпълнението си. Резервният компонент по принцип е хард диск, който има бърз достъп и е достатъчно голям, за да си набави копия на всички изображения на паметта за потребителите.

40. ***Виртуална памет на процеса – функционално разделяне (програма, данни, стек, heap, споделени библиотеки).***

- Виртуална памет — системна памет, симулирана от операционната система и разположена на твърдия диск. Тя позволява да се прилага едно и също, непрекъснато адресиране на физически различни памети (участъци от твърдия диск). Разделя се на различни парчета:
    - Статична част (read only) – има блок с фиксирани размери, където се разполага кодът на програмата, памет, която няма да се променя, записана е самата програма, изчислява се размерът предварително при стартирането; има парчета за споделени библиотеки, които са достъпни едновременно за няколко процеса и се предоставят на процеса (пестене на място);
    - Динамична част (read-write)
        - Стекове – намира към края на адресното пространство; нужен на процеса за управление на програмата; в него съществува реализацията на структурите за управление на програмата предоставени от езика за програмиране (функции, променливи, подпрограми); проста структура, заема обем от паметта, който може да нараства и да намалява.
        - Heap – памет, която се използва от процеса, ако е необходимо да се задели допълнително обеми памет, които не са в локалните променливи (използва се при нужда и тя варира от процес до процес, блок с неопределена дължина)
        - Блок за статична памет – променливи, които при стартиране на процеса се заделя винаги тази памет; място за данни-обем на масиви, константи и др.

41. ***Таблици за съответствието виртуална/реална памет.***

- Хардуерът, който отговаря за транслацията между логически и физически номер на страницата и въобще за управлението на адресацията, се нарича MMU (Memory Management Unit). Инструментът, който използваме за оптимизиране на работата на MMU, се нарича TLB (Translation Lookaside Buffer). TLB има фиксиран брой слотове, съдържащи записи от таблицата със страници и от таблицата със сегменти; Записите от таблицата със страници се използват за преобразуване на виртуалните адреси във физически адреси, а записите от таблицата със сегменти – за преобразуване на виртуалните адреси в сегментни адреси.

42. ***Файлови дескриптори, Номера на стандартните fd, Пренасочване***

- Файловият дескриптор – при отваряне на файл, ядрото извършва необходимите действия за отваряне на файла, зарежда информация за него. След това връща файловия дескриптор (неотрицателно цяло число, което служи за уникален идентификатор на файла), който се свързва с отворения файл, ползва се в програмата, а след затваряне се

освобождава. Дескрипторът има локално значение - връзката между дескриптор и отворен файл важи само за текущия процес.

- Операциите за вход, изход и грешка имат стандартни файлови дескриптори (0 – stdin, 1 – stdout и 2 – stderr).
- | - прави композиция на две програми; p1 | p2 изходните данни от първата се подават като входни на втората (пренасочва се stdout-ът на първата програма към stdin-а на втората програма). Първоначално, shell-ът казва на ядрото да създаде pipe (тръба). След това шелът (shell) започва да се размножава (fork() – системно извикване, което прави копие на извикващия процес). И двата процеса изпълняват една и съща програма, но всеки от тях работи в собствената си памет. Новият процес (детето) наследява всичко от стария (родителя). И така, след fork-а, имаме два shell-а, два процеса, които имат еднакви файлови дескриптори, еднакво съдържание на паметта. Единственото, по което може да се разбере кой е родителят и кой – наследникът, е резултатът, който е върнал fork() - при 0, наследник, при !=0, родител.

43. ***Опишете накратко стандарта RAID5. Какво е журнална файлова система?***

- RAID – Redundant Array of Independent/Inexpensive disk - стандарт който осигурява защита на данните, така че ако един диск изгори, информацията да не се загуби. При него дисковете се разделят на сектори. За всеки сектор, контролния диск се изчислява като номер на сектор по модул броя дискове, например

- За първи сектор, контролния диск го избираме на първия диск, за втори сектор на втория и така ги редуваме, като записа става равномерно и дисковете ще се изтъркват равномерно (за разлика от RAID4).

44. ***Свързване и допускане до UNIX система – login.***

- The login program is used to establish a new session with the system. It is normally invoked automatically by responding to the "login:" prompt on the user's terminal. login may be special to the shell and may not be invoked as a sub-process. When called from a shell, login should be executed as exec login which will cause the user to exit from the current shell (and thus will prevent the new logged in user to return to the session of the caller). Attempting to execute login from any shell but the login shell will produce an error message.

45. ***Команден интерпретатор – shell. Изпълнение на команди, параметри на команди.***

- Когато включим конзолата (след като сме се идентифицирали), стартира програма shell. Тя има много различни реализации (bash, zsh, tcsh, sh). Редът преди курсора се нарича prompt и ни казва в какъв режим работим (обикновен потребител, root).
- Най-общо има два вида команди – команди-филтри и команди, които показват състоянието на системата.
- Основни команди:
  - df - (disc free) файловите системи
  - ps - показва текущите процеси
  - ps -ef - показва всички процеси
  - wc -l - брой редове
  - cut – вади дадени колони от изхода
  - uname - unix name

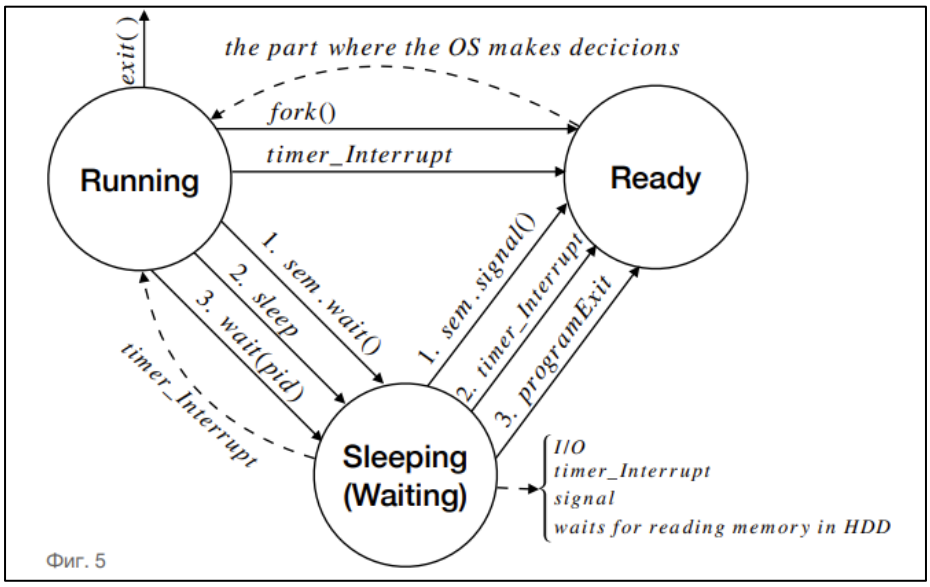
46. ***Файлове и директории***

- В UNIX абстрактната файлова система е кореново дърво. Коренът представя началото на файловата система. Върховете представляват директории, а листата - файлове, като всеки връх (директория) също представлява кореново дърво. Към всеки файл, освен името и съдържанието, се асоциират и други атрибути – тип на файла ("-" – обикновен файл, "d" – директория, "c" – символно устройство/character special device (напр. клавиатура), "b" – блоково устройство/block special device (външно устройство, което съхранява масив от байтове), "p" – именувана тръба (FIFO), "s" – socket/конектор (крайна точка за комуникация)), права за достъп, потребител-собственик, потребителска група, размер, дата на промяна, дата на създаване и др.

47. ***Възможни състояния на процес.***

- Running – процеси, които активно се нуждаят и използват ядро и процесор, който изчислява. Това са процеси, които от гледна точка на потребителя имат работа за вършене. Те може да се изчакват и редуват, ако не стигат процесорите, но като цяло имат нужда от изчислителна мощ. От гледна точка на потебителя те са една група, но от гледна точка на ядрото те са:
    - Running – изчисляват се в момента;
    - Ready – в момента няма процесор за тях, но при следващия такт те ще получат управление.
- Sleeping – спящите процеси от гледна точка на потребителя са работещи програми, които са стигнали в състояние, при което нямат нужда да изчисляват нещо докато не настъпят интересни за тях събития в системата. Те са в комуникация с друг процес или устройство и очакват да им се подадат данни, но очакваните процеси нямат готовност да им подадат (например поради запушен комуникационен канал или поради бавна работа на другата страна и т.н.). От гледна точка на реализацията може да чакат:
    - I/O – очаква извършване на входно изходни операции;
    - Time – очаква да настъпи времеви момент;
    - Signal - очаква сигнал от друг процес за промяна на състоянието му (на другия процес);
    - Процеса бива приспан, защото страницата, с която иска да работи не е на реалната памет, а е някъде на твърдия диск.
- Stopped – спрян процес (не представлява интерес нито за потребителите, нито за операционната система)
- Zombie – процес, при който е започнало спирането но не е завършило (пускането и спирането на процес са бавни и многостъпкови събития)

48. <u>Диаграма на състоянията и преходите между тях.</u>



49. <u>Избройте видове събития, причиняващи повреда на данните във файловите системи.</u>

- Хардуерни причини – паметта, която ни предоставя хардуерът, е някакво физическо устойство, но тези устройства могат да дефектират (да остареят, да се счупят); могат да дефектират по икономически причини (създава се хардуер с по-ниско качество, за да е по-ниска цената) и т.н.
- Софтуерни причини – бъг в операционната система, който се проявява при определени условия; например, ако операционната система не е добре написана е възможно някъде да възникне deadlock или starvation
- Потребителска грешка – поради невнимание, липса на знание или други причини е възможно да настъпи загуба на данните (напр. Изтриване), предизвикано от потребителя; потребителските грешки могат да зависят и от потребителския интерфейс (например, в конзолния интерфейс проста синтактична грешка може да доведе до големи проблеми (rm *.txt и rm * .txt))
- Недоброжелателни агенти – вируси, хакери и т.н. ; съществен проблем при големи и важни системи.
- Стандарти – остаряване на стандарти и абстракции; честа смяна на стандартите; различни кодировки.
- Аварийни ситуации – спиране на захранване, наводнение и т.н.

50. <u>„Пространство на имената“. Как изглежда това пространство в ОС Linux?</u>

- Пространство на имената – набор от “имена”, които се използват за идентифициране и препращане към обекти от различен вид; гарантира, че всички дадени групи обекти имат уникални имена, така че да могат лесно да бъдат идентифицирани; това са всички дълготрайни обекти, които съществуват в системата – абстрактната файлова система;

51. <u>Конзола – стандартен вход, стандартен изход, стандартна грешка.</u>

- Стандартните потоци са вход и изход – комуникационните канали между програма и нейното обкръжение, когато започне да се изпълнява. Трите I/O връзки се наричат стандартен вход (stdin), стандартен изход (stdout) и стандартна грешка (stderr). Когато команда е изпълнена през интерактивен shell, потоците са свързани към текстовия терминал, на който работи shell-а, но може да се промени чрез пренасочване (pipeline). По общо казано, child process ще наследи стандартните потоци на родителя си.

Ден 3. (02.07.2021 г.)

Ами в този ден или си се отчаял и си си теглил куршума, или си тотално откачил и не си загубил надеждата в себе си, че може и да успееш.

52. Качества и свойства на конкретните файлови системи, реализирани върху block devices.

- В UNIX е прието в директориите да се съдържат прости обекти, които се състоят от <име, inode>. Inode идва от index node – номер на блок върху диска, по който може да разберем всички атрибути на файла. Под име се приема абсолютен път към наследника. Тази особеност на UNIX позволява един и същи файл да има няколко имена (hardlinks). По този начин, един файл ще бъде изтрит, чак когато се изтрият всички негови твърди връзки. При всяко отрваряне на файл се създава нова твърда връзка, което дава възможност на програмата да продължи своята работа, дори ако друг процес промени съдържанието на файла.
- Inode е указател към таблица, която се състои от 2 части: атрибути и указатели (адресна таблица); например, ако в адресната таблица има 13 указателя, първите 10 са директни адреси към първите 10 сектора на файла; 11ти,12ти указател са прости таблици, т.е. сектори, които съдържат адреси на сектори; 13 указател е таблица с 3 слоя (сочи към таблица, изградена от таблици, които съдържат сектори)
- Самият твърд диск се състои от няколко области: група inodes (които от своя страна са разделени на отделни области), по-голяма област със сектори, които съдържат метаданни и две скужебни зони, в които да се запише информация за това кои сектори са свободни и кои inodes са свободни.

inodes	sectors	i	s

53. Механизми и структури за приспиване/събуждане.

54. Поддържане на буфери (кеширане) на файловата система.

За да се ускори работата на системата се пазят специални кешове, в който се пазят най-често използваните редове от адресната таблица. Има 3 вида кешове:

- Кеш за инструкции - изпълнимия код се разполага в специален кеш, който поддържа сравнително дълги парчета от RAM паметта и сравнително малко на брой;
- Кеш за работа с данни - къси фрагменти ще се използват много често (напр. броячи, локални променливи); четене и писане на данни
- Кеш за упавление на виртуална памет – използва се т.нар

TLB – Translation lookaside buffer

- Кешове за няколко отделни таблици за управление на виртуална памет – съдържа 32-битови думи
- Няма голям размер
- Има две нива

  При процесорите делим паметта между тях и даваме на всеки толкова, колкото му трябва, но те са анонимни, те са парчета информация. При файлове парчетата от памет трябва да са именувани и централния въпрос е за имената. Когато файла пази информация трябва да е в устойчивата памет. Когато променяме информация във файла, той трябва да е свързан с процес

  Паметта може да се представи като множество от файлове – дълготрайано живеещи в информационната среда – информационни обекти. Тъй като файловете ще съществуват дълго са важни имената им.