**Системни примитиви**

open() – По подадено име на файл и флагове, връща файлов дескриптор на отворения файл. Ако такъв файл не съществува, при подаден флаг O\_CREAT се създава. При грешка връща -1.

close() – Затваря подаден файлов дескриптор. При грешка връща -1.

read() – Чете брой байтове от файлов дескриптор и ги записва в подаден буфер (адрес в паметта). Връща брой прочетени байтове или -1 при грешка.

write() – Пише брой байтове от буфер (адрес в паметта) във файлов дескриптор. Връща брой записани байтове или -1 при грешка.

lseek() – Позиционира курсора на подадения файлов дескриптор на n-байта спрямо началото/сегашната позиция/края на файла.

pipe() – Взима масив от два int, които ги инициализира съответно на дескриптор за писане и четене, т.е. създава комуникационен канал тръба между процеси. При успех връща 0, при грешка връща -1.

dup2() – Приема два файлови дескриптора, стар и нов, създава копие на стария дескриптор със стойността на новия дескриптор. Ако новия дескриптор вече съществува, автоматично се затваря. При успех връща новия дескриптор, при грешка връща -1.

fork() – Създава нов процес, който е копие на извикващия fork() процес. Двата процеса работят паралелно в различни области от паметта. При успех връща PID на детето при родителя и 0 при детето, при грешка връща -1.

exec() – Заменя текущата процесна обработка с нова такава. Само при грешка връща -1.

wait() – Прекъсва изпълнението на текущия процес докато едно от децата му се терминира. При успех връща PID на терминираното дете, при грешка връща -1.

waitpid() – Прекъсва изпълнението на текущия процес, докато детето, съответстващо на подадения PID, промени състоянието си (по подразбиране е неговото терминиране, има опции за промяна на това държание). При успех връща PID на измененото дете, при грешка връща -1.

**Bash инструменти**

chmod – променя правата за достъп на файлове/каталози

cd – сменя текущата директория

mkdir – създава директории с указаните имена, ако такива не съществуват

rmdir – изтрива указаните каталози, ако такива съществуват и са празни

cp – прави копие на файл/каталог

mv – мести файл/каталог

rm – изтрива указаните файлове, т.е. намалява твърдите връзки на съответните inode-ове

ls – извежда името на файл или ако е каталог – извежда неговото съдържание

who – извежда кои потребители в момента са в сесия

find – търси файлове в йерархията на директория

ps – извежда информация за съществуващите процеси в системата; по подразбиране връща информация само за процесите, които принадлежат на потребителя, изпълняващ ps

top – извежда информация за текущите процеси в реално време (подобно на task manager в Windows)

vi – визуален текстов редактор

tar – инструмент за архивиране

gcc – GNU компилатор за C и C++

echo – извежда на стандартния изход своя вход

read – чете позиционни параметри от стандартния вход, разделени с интервал, и ги записва в съответни променливи ${n}, където n = 1,2,3… пореден номер на позиционния параметър

test – проверява условие и връща код на завършване 0, ако е изпълнено

if – условен оператор, ако условието е вярно се изпълнява тялото

for – цикъл, въртящ се докато е изпълнено някакво условие, позволява създаване на локална променлива и обработка на тази променлива

while – цикъл, въртящ се докато е изпълнено някакво условие

cat – извежда на стандартния изход съдържанието на аргументите си или ако няма такива – на стандартния вход

grep – търси даден шаблон във всеки от указаните файлове, като извежда редовете, където той е открит

cut – извежда определена част от всеки ред на всеки от указаните файлове

sort – слива съдържанието на указаните файлове и го сортира лексикографски по редове, като извежда резултата на стандартния изход

wc – извежда на стандартния изход броя на байтовете (символите), думите и редовете във всички указани файлове поотделно

tr – заменя или изтрива указани символи

ln – създава твърда връзка към даден файл, -s за символна връзка

**Linux директории**

/etc – съдържа конфигурационни файлове на системата, които контролират държанието на операционната система или приложенията

/dev – съдържа псевдофайлове на устройствата, управлявани от операционната система

/var – съдържа файлове/данни, които се изменят с времето, типичен пример лог-файлове

/var/log – съдържа log файлове

/proc – съдържа статистическа информация за ядрото, чрез нея програмите могат да видят какво става в ядрото, кои процеси са активни, кой с кой си говори, кои файлове са отворени, кой файл в кой процес се ползва, връзките в Интернет

/bin – съдържа бинарни и изпълними файлове, общи за всички потребители

/sbin – системни програми, използвани от root при зареждането на ОС

/home – съдържа поддиректориите за всички потребители

/usr – съдържа всички изпълними файлове, библиотеки, програми за потребителите

/usr/bin – съдържа основни потребителски команди

/usr/sbin – съдържа допълнителни команди за администратора

/usr/doc – ?

/usr/lib – съдържа системните библиотеки

/boot – съдържа файлове, които отговарят за зареждане на операционната система, преди ядрото да е тръгнало да работи

/tmp – съдържа временни файлове

/root – домашната директория на суперпотребителя

**Въпроси**

1. **Опишете как ОС разделят ресурсите на изчислителната система, дайте примери за основните типове разделяне:**
   1. **Разделяне на пространството (памети)**
   2. **Разделяне на времето (процесори, други у-ва)**

а) Ресурса 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 и справедливо да разделя ресурса. Ресурси управлявани по този начин, са оперативна и дискова памет.

б) Програмите или потребителите (процесите) използват ресурса последователно едни след други. Задачата на ОС е да решава: кой да е следващия, колко дълго да го използва, да отнема ли насилствено ресурс. Когато ОС насилствено отнема ресурс от процес, казваме че има преразпределение. Примери: централен процесор (обикновено с преразпределение), печатащо устройство (обикновено без преразпределение).

1. **Опишете разликата между времеделене и многозадачност. Какви ресурси разделя еднозадачна, еднопотребителска ОС?**

Еднозначната еднопотребителска ОС заделя всички ресурси за този един потребител и тази една задача. В действителност има такава операционна система. Ако процесорът е един, т.е. само един процес, то времеделението, това е квант време, т.е. неделимо малко време, което може да се предостави на един процес за изпълнение, след това се дава на друг процес, при което може за 1 секунда много задачи да се обиколят. Т.е. многозадачност чрез времеделение. Ако има многопроцесорна система или многоядрена, то би могло да се изпълняват паралелно толкова процеси, колкото е броят на процесорите, но операционната система трябва да поддържа синхронизацията.

1. **Дайте кратко определение за: многозадачна ОС, многопотребителска ОС, времеделене. Опишете разликата между многопотребителска и многозадачна работа.**

Многозадачна ОС – изпълнява няколко задачи едновременно. това се постига чрез разделяне на времето на работа на процесора според инструкциите на специална подсистема (scheduler).

Многопотребителска ОС – разширява концепцията за многозадачност като различава потребителите по отношение на ползването на процеси и ресурси, като например дисково пространство.

Времеделене – при наличие на един процесор е необходимо да се работи в режим на времеделене, който нарязва на малки интервали от време, в които да се изпълняват отделните процеси, превключвани чрез таймера.

1. **Опишете ситуацията съревнование за ресурси (race condition), дайте пример. Опишете накратко инструментите за избягване на race condition:**
   1. **дефинирайте критична секция, атомарна обработка на ресурса;.**
   2. **инстурменти от ниско ниво, специфични хардуерни средства;**
   3. **инструменти от високо ниво, които блокират и събуждат процес.**

Ситуация, при която два или повече процеса използват общ ресурс, се нарича *race condition*. Пример: обща променлива *counter* и два процеса, които четат и пишат в нея.

а) Критична секция – част от кода на процеса, в която се осъществява достъп до обща памет или върши неща, които могат да доведат до състезание. Атомарна обработка на ресурса означава, че процесите няма да си пречат едни на други при изпълнение на критичната секция.

б) Спиране на прекъсванията в еднопроцесорна система; mutex, който представлява един бит: test-and-set – атомарна размяна на стойността от регистър и клетка памет; atomic swap – сравнява съдържанието на локация в паметта с дадена стойност и само ако са същите, модифицира съдържанието с новата стойност; spinlock – ключ, който кара нишката, която се опитва да го вземе, да изчаква в цикъл, проверявайки постоянно дали ключа е свободен, използва ненужно ресурси.

в) За синхронизация от високо ниво се използват семафори, които се използват за приспиване и събуждане на процес.

1. **Хардуерни инструменти за защита (lock) на ресурс:**
   1. **enable/disable interrupt**
   2. **test and set**
   3. **atomic swap**

а) Преди навлизане в критична секция, процесът може да спре прекъсванията като може да приключи своята работа без някой друг процес да му пречи. Това не върви в многопроцесорна система.

б) Инструкция, която чете съдържанието на адрес от регистъра и записва ненулева стойност на този адрес. Гарантирано е, че това се извършва като една независима операция. Така процесорът заключва паметната шина, предотвратявайки достъпа на други процесори към паметта.

в) Атомично сравнява стойността на адрес в паметта с подадена стойност и само ако съвпадат променя стойността в паметта с нова такава.

**Опишете инструмента spinlock, неговите предимства и недостатъци.**

Spinlock представлява ключ, който кара нишката, която се опитва да го вземе, да изчаква в цикъл, проверявайки постоянно дали ключа е свободен. Лесен е за имплементация, но предизвиква излишно ползване на ресурси, тъй като нишката остава активна (busy waiting).

1. **Опишете понятията приспиване и събуждане на процес (block/wakeup). Семафор – дефиниция и реализация. Опишете разликата между слаб и силен семафор.**

Когато процесор не може да навлезне в критичната си секция, той бива приспан, за да не губи процесорно време докато чака. Когато критичната секция е свободна, бива събуден от друг процес.

Семафор – механизъм за междупроцесни комуникации. С всеки семафор се свързва цяла променлива – брояч и контейнер на чакащите блокирани процеси. Когато структурата за чакащите процеси е обикновена опашка, казваме че семафорът е силен, т.е. позволява стого определен ред на предоставяне на ресурса. В противен случай няма ред, в който се събуждат процеси, т.е. имаме слаб семафор.

**Опишете накратко различните видове специални файлове в Linux: външни устройства, именувани в /dev, псевдофайлове в /proc, линкове – твърди и символни, команда ln, сокети**

Специалните файлове за външни устройства се делят на два вида – block special files (съдържат поредица от адресируеми блокове, обикновено се използват за дискове) и character special files (по принцип се използват за устройства, които пишат и отпечатват поток от символи/байтове).

Псевдофайловете в /proc се създават от процесите в системата. Пример: pipe, която позволява на два процеса да комуникират.

Твърдите линкове представляват различни наименования на един и същ файл. По този начин един и същ файл може да се появява в множество каталози. Символните линкове от своя страна сочат към малък файл, който е наименование на друг файл. По тях системата проследява пътя към крайното сочено име на файл. За създаване на твърди (символни) връзки се използва команда ln (с флаг -s).

Сокетите са двупосочни комуникационни канали, позволяващи комуникацията между процеси на различни системи (клиент-сървър).

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

За постигане на взаимно изключване може да се използва така наречения *mutex*. Необходим ни е само едни семафор, инициализиран с 1. Процесите трябва да заградят критичната си секция между wait() и signal() операциите на този семафор.

**Качества и свойства на конкретните файлови системи, реализирани върху block devices. Ефективна реализация, отлагане на записа, алгоритъм на асансьора.**

Файловите системи трябва да осигуряват бърз достъп до данните и ефективно използване на дисковата памет, да са устойчиви в условията на конкурентен достъп от много потребители, при възможен срив да са защитени от несанкциониран достъп.

За ефективна реализация на ФС искаме прости абстракции, а не прекалено сложни концепции. В съвременната файлова система таблицата на адресите от сектори, изграждащи един файл, може да се реализира като дърво. Друга реализация включва битова карта, която индикира кои блокове са свободни и кои не. Свободните блокове във файловата система трябва да са организирани по някакъв начин за ефективен достъп, едно решение за това е използване на свързан списък.

За добра производителност се използва кеширане на често използвани данни от диска, като се заделя част от RAM-паметта за входно/изходни буфери. Ако много често презаписваме едни и същи сектори от данни, може да се използва т.нар. отлагане на записа, т.е. да се изчаква малко преди информацията от кеша да се отрази върху самия диск. Освен това можем да пренаредим така записите, че главата на диска да изминава минимално разстояние. При алгоритъма на асансьора главата на диска се движи в една посока, докато не свършат вече операциите по секторите в тази посока.

1. **Права и роли в UNIX, команда chmod. Права – u/g/o user/group/other. Роли – r/w/x read/write/execute**

Ако изчислителната система има много потребители, трябва да се поддържат свойства, които описват правата и привилегиите на потребителите за даден ресурс. Права и роли – всеки файл си има код за защита, на който се падат 12 от битовете на mode на inode-а. За всяка категория u/g/o имаме трите роли r/w/x, които се закодират в 3 от четирите им предоставени бита. Останалия един бит отговаря за съответно SUID/SGID/Sticky bit.

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

Комуникационния канал тръба между два процеса може да се реализира чрез семафори, които да позволят на производителя (този, който пише в тръбата) и потребителя (този, който чете от тръбата) да се редуват за достъп до споделения ресурс, т.е. данните в тръбата, като първо разбира се трябва да мине производителя, тъй като в началото тръбата ще бъде празна. След което потребителят ще може да прочете данните от тръбата и така да освободи мястото. Пример за подобна комуникация е изпращането и получаването на пакети в Интернет.

1. **Взаимно блокиране (deadlock). Гладуване (livelock, resource starvation). Пример: задача за философите и макароните**

Deadlock е ситуация, в която процес блокира, за да изчака някакво събитие, което да го събуди, но такова събитие никога не се случва. Starvation е ситуация, в която процес може да бъде приспан за неопределено дълго време, било то заради използване на слаби семафори или неподходяща реализация на приоритетна опашка.

В задачата за философите и макароните имаме 5 философа седнали на една кръгла маса, на която има чиния за всеки философ и вилица между всеки двама. Всеки философ изпълнява следните функции: мисли, взима вилици и яде, като за да яде са му нужни и лявата, и дясната вилица. Трябва да са изпълнени следните изисквания за синхронизация: 1) само 1 философ може да притежава конкретна вилица във всеки един момент; 2) трябва да не може да настъпи deadlock; 3) трябва да не е възможно философ да гладува; 4) трябва да може повече от един философ да се храни.

**Единна йерархична файлова система в UNIX. Файлове и директории, команди – cd, mkdir, rmdir, cp, mv, rm**

Файловата система в UNIX, подобно на повечето съвременни ФС, представлява едно кореново дърво, чийто корен е директорията /, а в нея се намират всички други поддиректории и файлове. Когато дисковото пространство се разделя на части, тези части стават поддървета на един главен диск, който операционната система първо вижда, разглежда и използва при своето зареждане. Той винаги е закачен към корена на файловата система, а към самия него могат да се монтират други дискове.

1. **Процеси в многозадачната система. Превключване, управлявано от синхронизация. Превключване в система с времеделене – timer interrupt.**

В многозадачна система имаме множество процеси, които се съревновават за процесорно време. Тези процеси се делят на няколко групи според функционалността си:

I/O – основно или често извършващи входно/изходни операции (*interactive*, *foreground*). За такива процеси е характерно, че ще правят малко изчисления, например чакат някакъв текст и го изпращат някъде. Но те често ще бъдат приспивани и често ще прекарват повече време в другите структури между Running и Ready. Затова искаме бързо да ги вкараме в активно състояние, за да има ниска латентност.

CPU – такива, които смятат повече (*background*). За тях важното е да им се предостави много изчислително време, но не е толкова важно кога

Real-time – при тях има изисквания, които са доста по-строги. Това са приложения, които ще комуникират с околния свят – т.е. управляват някакви важни машини, устройства или инструменти и е критично важно да няма забавяне, спрямо предварително изграден времеви план. Ти имат т.нар. *deadline* – предварително дефинирани срокове, за които да се получи управлението на системата.

Най-простия алгоритъм с времеделене е *Round Robin*, който нарежда чакащите процеси в обикновена опашка. Така няма никакви приоритети и няма да има гладуване на процеси, но както казахме реално някои процеси имат изисквания. Това може да се постигне с приоритетна опашка и за да няма гладуване, може да се въведат т.нар. плаващи приоритети.

1. **Възможни състояния на процес. Механизми и структури за приспиване/събуждане. Диаграма на състоянията и преходите между тях.**

При стартирането си всеки процес вече има две основни състояния: *running* – процесът в момента има нужда от процесорно време, *sleeping* – процесът няма нужда от процесорно време, защото очаква някакво събитие. Реалната бройка на изпълняващите се процеси е фиксирана, т.е. в едно време могат да се изпълняват *k* процеси, а останалите стоят в състояние *ready.*

Процесите могат да бъдат приспивани от други процеси чрез сигнали, от самосебе си или от някакви входно/изходни операции. За процеси, които чакат да получат процесорно време, има приоритетна опашка, която помага за определяне кой процес да получи ресурс. Това може да предизвика гладуване, затова се въвеждат плаващи приоритети, т.е. един процес ако поиска да върши работа, но не му се предостави процесорно време, му се вдига приоритета и евентуално ще го получи.

**Опишете функционалността на следните команди в Linux: vi, tar, gcc**

vi – текстов редактор, предоставящ най-различни функционалности за обработка

tar – инструмент за архивиране/разархивиране на файлове

gcc – инструмент за компилиране на C/C++ код

1. **Процес и неговата локална памет – методи за изолация и защита. Йерархия на паметите – кеш, RAM, swap. Виртуална памет на процеса – функционално разделяне (програма, данни, стек, heap, споделени библиотеки).**

Проста техника за изолация на локалната памер е сегментацията: за паметта, която е предоставена на всеки процес имаме два регистъра: начало (*f*) и дължина (*s*) и проверяваме адрес на всеки байг, който използва този процес дали е в този интервал, т.е. *f <= address <= f+s*. Друга техника е *paging* на *IBM 360*: имаме *n* страници и всяка страница има атрибути, в които е записано как и кои процеси могат да използват тази страница. В съвременната ОС аналог на *paging* е виртуалната памет.

На най-високо ниво работят процесорните регистри, които са бърза, но малка по обем памет. Под тях се намират кешове на самия чип, след което може да има други нива на кеш извън чипа. Накрая стои RAM-паметта. Част от страниците на процесите, които са пасивни, може да се разполагат и в swap пространството от твърдия диск, което се използва за разширяване на RAM-паметта.

На всеки процес се предоставя памет, която логически изглежда като реалната памет, която се нарича виртуална. За процеса паметта му изглежда като едно линейно пространство, единствен масив от RAM, разделен на страници. В съвременните системи това позволява еднообразна подготовка на програмите. Компилаторите могат този еднообразен масив да си го подготвят под еднообразен начин – в началото стои кода на програмата, в края стои стекът, след кода се слагат глобалните данни на процеса (локалните се пазят в стека), останалото пространство се ползва за heap-а и също някакъв блок за споделените библиотеки.

1. **Таблици за съответствието виртуална/реална памет. Ефективна обработка на адресацията – MMU, TLB.**

На всеки процес се предоставя памет, която логически изглежда като реалната памет, която се нарича виртуална. За процеса паметта му изглежда като едно линейно пространство, единствен масив от RAM, разделен на страници, които не съответстват на страници от реалната памет, а се използват таблици, съдържащи съответствията на виртуална/реална памет и също дали могат да се ползват от процеса.

MMU (Memory management unit) е хардуерното устройство, което отговаря за проследяване на адреса от локалната памет на процеса, прочитане на данните от адресната таблица на процеса, правене на заместване и извършване на истинската адресация.

TLB (Translation lookaside buffer) е специален кеш, който позволява бързото достъпване на често използвани адресни таблици. Този кеш е хубаво да е асоциативен, тоест всеки ред или всяка област от редове от кеша сами да проверяват за съвпадение на адрес.

**Файлови дескриптори, номера на стандартните fd, пренасочване, филтри – cat, grep, cut, sort, wc, tr**

Всеки процес има на разположение три стандартно отворени файла, които може да използва: стандартен вход, стандартен изход и стандартен изход за грешки. И трите файла са свързани по подразбиране с терминала на потребителя. С всеки отворен файл се свързва файлов дескриптор, който е цяло неотрицателно число и посредством него се идентифицира отворения файл в системните примитиви, изпълнявани от процеса. Файловите дескриптори на стандартните файлове са съответно 0, 1 и 2. Преди да изпълни определена команда shell може да свърже стандартните файлове с нещо различно от терминала и това се нарича пренасочване на вход/изход. Синтаксисът за пренасочването:

команда <файл – пренасочва стандартния вход на командата от файла

команда [12]>файл – пренасочва стандартния изход/изход за грешки на командата към файла, ако файлът не съществува, първо се създава, а иначе старото му съдържание се изтрива

команда [12]>>файл – пренасочва стандартния изход/изход за грешки на командата към файла, ако файлът не съществува, първо се създава, а иначе се отваря за добавяне в края му

1. **Избройте видове събития, причиняващи повреда на данните във файловите системи. Опишете накратко стандарта RAID5. Какво е журнална файлова система?**

Първата важна причина за загуба на информация е проблем с хардуера (остаряване, счупване, външни влияния). Друга причина е софтуерна грешка, т.е. в самите инструменти за предоставяне на абстракция (бъг в ядрото). Също така, потребителят или потребителския интерфейс може да се окаже причинител за загуба на информация било то поради неумело използване на софтуерните инструменти. Други причини може да са хакерски атаки, остаряване на стандарти и абстракции, спиране на тока.

При RAID5 стандарта дисковете са разделени на сектори и за всеки сектор контролния диск се смята <номер на сектор>%<брой дискове>. Така равномерно ще се разпределя товара между дисковете за разлика от RAID4, където само в един контролен диск става това.

В журнална файлова система имаме структура, която пази какви транзакции ще бъдат извършени върху файловете. Тази структура се нарича журнал, представляващ таблица на транзакциите. Когато този журнал се запълни, започват да се отразяват първите транзакции, записани в него, на самия диск.

**Свързване и допускане до UNIX система – login. Конзола – стандартен вход, стандартен изход, стандартна грешка. Команден интерпретатор – shell. Изпълнение на команди, параметри на команди.**

След като потребителят успешно преодолее проверките по идентификацията си, за него автоматично се създава процес с програмата-shell. Този процес се нарича login-shell процес. Първото му действие е да инициализира обкръжението си, след това влиза в цикъл, в който извежда покана, и чете и изпълнява командите на потребителя. Покана (*prompt*) е съобщение, извеждано от процеса-shell, с което заявява готовността си да приеме нова команда.

В командния език има команда (logout или exit), която означава край на работата на потребителя със системата. Диалогът между потребителя и системата от идентификацията (login) до изпълнение на командата logout наричаме сесия.

Изпълнението на команда става чрез командни процедури (команден файл, shell файл, shell script), които са текстови файлове, съдържащи конструкции на командния език на shell, следователно това е програма на командния език, която се интерпретира от shell. В командната процедура могат да се използват позиционни параметри.

1. **Опишете разликата между синхронни и асинхронни входно-изходни операции. Дайте примери за програми, при които се налага използването на асинхронен вход-изход.**

При синхронна операция процеса може да се блокира, да се приспи и операциите се извършват в тяхната цялост. Ако операционната система може да окомплектова операцията – то я окомплектова и тогава връща резултат, иначе връща код за грешка по някаква друга причина.

При асинхронна операция процеса не се приспива, но това се заплаща с цената на върнат на части резултат от операцията и това се индикира със специален код за грешка, който показва, че процеса трябва да се довършва. Потребителят сам се грижи да прави последващи извиквания 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.

За примери можем да дадем уеб-браузъра, той трябва да реагира на входни данни от клавиатурата и мишката, както и на данните, постъпващи от интернет, т.е. на поне 3 входни канала.

1. **Опишете понятието „пространство на имената“. Как изглежда това пространство в ОС Linux?**

Пространството на имената е абстрактна файлова система. То е множеството от всички дълготрайни обекти, които съществуват в системата. В Linux всяко име е асоциирано с inode, който съдържа всички данни за един файл, т.е. с всеки файл се асоциира един inode, но един файл може да има различни наименования (hardlinks).

**Една от класическите задачи за синхронизация се нарича „Задача за читателите и писателите“ (readers-writers problem). Опишете условието на задачата и решение, използващо семафори.**

Виж файла с 20-те въпроса.

1. **Опишете какви атрибути имат файловете в съвременна файлова система, реализирана върху блочно устройство (block device). Опишете накратко реализацията и целта на следните инструменти:**
   1. **отлагане на записа, алгоритъм на асансьора;**
   2. **поддържане на буфери (кеширане) на файловата система.**

а) Главата, която върви по твърдия диск и записва, върви от първи сектор до последен и намира най-близката заявка за запис от I/O кеша, като стигне до последния, тръгва наобратно и повтаря търсенето сред заявките. Това гарантира, че няма да има starvation, който може да се получи с приоритетна опашка

б) Кеширането позволява по-добра производителност при изпълнение на повтарящи се заявки.

**Опишете как се изгражда комуникационен канал (connection) между процес-сървър и процес-клиент със следните системни изисквания в стандарта POSIX: socket(), bind(), connect(), listen(), accept()**

Сървърът първо създава сокет чрез sfd=socket(domain, type, protocol), присвоява име на сокета – bind(sfd, &my\_addr, addrlen), започва да приема заявки за връзка чрез listen(sfd, backlog) и накрая приема заявка за изграждане на връзка cfd=accept(sfd,, &peer\_addr, addrlen).

Клиентът от своя страна създава сокет fd=socket(domain, type, protocol) и подава заявка за изграждане на връзка connect(fd, &server\_addr, addrlen).

1. **Опишете накратко основните комуникационни канали в ОС Linux. Кои канали използват пространството на имената и кои не го правят?**

Основните ком. канали в Linux са неименувана тръба (pipе), именувана тръба (FIFO), връзка процес-файл и конекция (connection). Именуваната тръба и конекцията използват пространството на имената.

**Опишете какви изисквания удовлетворява съвременната файлова система, реализирана върху блочно устройство (block device). Опишете предназначението на журнала на файловата система.**

Съвременните файлови системи трябва да са ефективни (да осигуряват бърз достъп до данните и ефективно използване на дискова памет), надеждни и сигурни (да са устойчиви в условията на конкурентен достъп от много потребители, при възможни сривове да е защитена от несанкциониран достъп) и разширяеми (физическата организация трябва да е адекватна на съвременното състояние и тенденциите в развитието на хардуера).

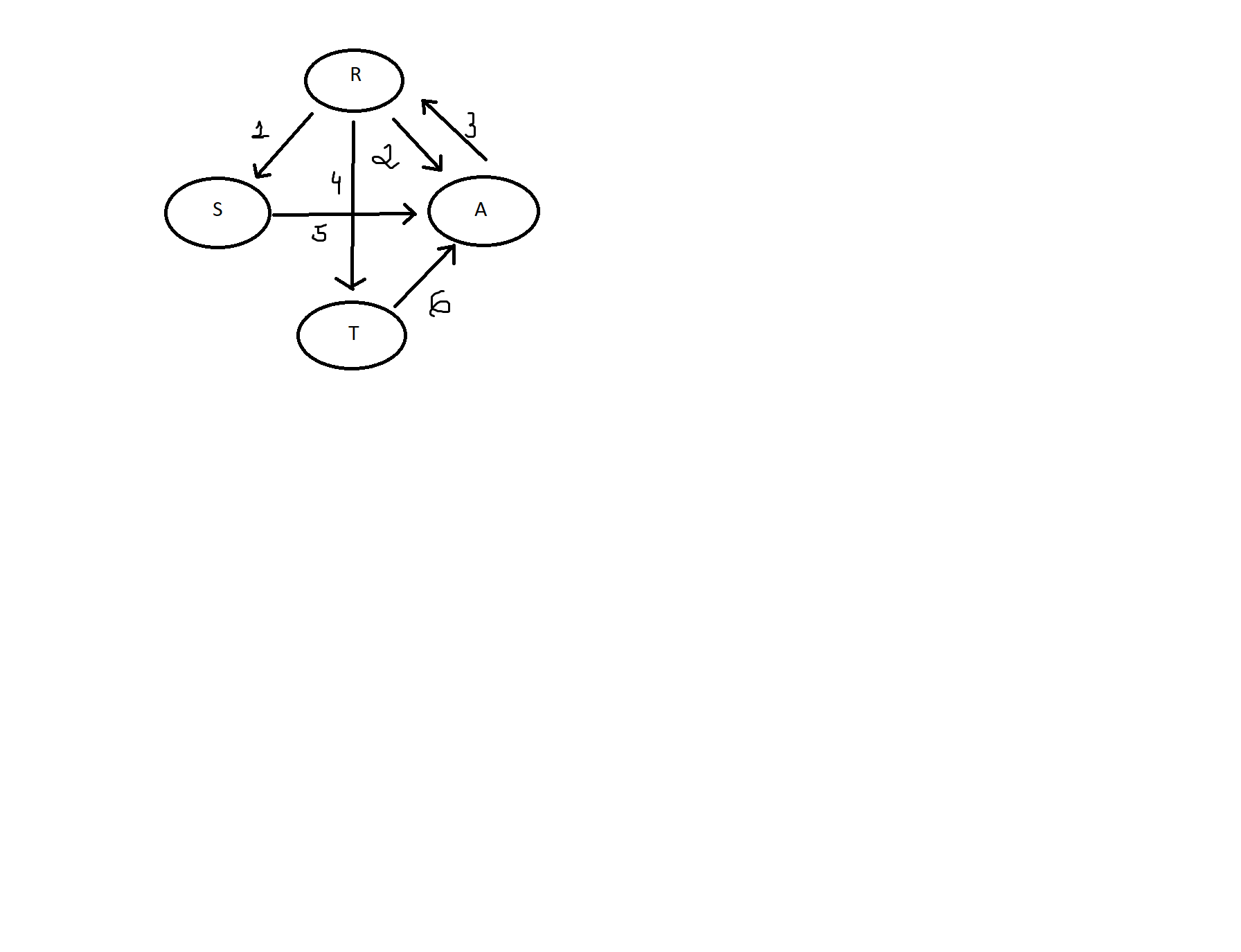
В журнала се записват метаданните за файловата операция. От там с алгоритъма на асансьора се прехвърлят на диска. И при ъпдейт на данните новите данни се записват на нов сектор – отбелязва се в журнала новия и стария сектор. Стария се освобождава. Така като спре тока оригиналните данни няма да са съсипани.

1. **Опишете максимално несправедлива ситуация, която може да се получи в избирателна секция, ако на входа на секцията пазач – член на изборната комисия, пуска гласоподавателите вътре така:**
   1. **във всеки момент в секцията може да има най-много двама гласоподаватели**
   2. **пазачът работи като слаб семафор**

Да речем, че пристигнат двама гласоподаватели, които са приятели на пазача. Той ги пуска вътре и след това пристига трети гласоподавател, който не е негов приятел и не му се харесва много. Третият започва да чака своя ред, но тъй като пазачът работи като слаб семафор и всички други пристигащи гласоподаватели са му приятели, с изключение на третия, този трети ще чака неопределено време и ще си умре там от глад.

1. **Да приемем, че в съвременната операционна система процесът има 4 състояния:**
   1. **R – работещ (*running*, използва CPU)**
   2. **A – активен (*ready*, очаква CPU)**
   3. **S – блокиран (*sleeping*, очаква вход/изход)**
   4. **T – изчакващ време (*sleeping*, очаква времеви момент)**

**Нарисувайте диаграма на състоянията и преходите между тях. Диаграмата е ориентиран граф с върхове отделните състояния и ребра – възможните преходи. Опишете накратко събитията, предизвикващи преход по всяко ребро на графа.**

1) Когато процес чете от тръба/специален файл и няма вход от данни.

2), 3) Предизвикват се от планировчика (process scheduler), без знанието на процесите. 2) Когато планировчика реши, че процесът се е изпълнявал достатъчно дълго. 3) Когато всички други процеси са си изпълнили своя дял от времето и е време да се предостави CPU на първия готов процес.

4) Когато операционна система реши, че процес не може да продължи работата си.

5) и 6) Когато външно събитие, за което процес чака, се случи.

1. **Процесът P създава тръба (pipe) с извикване на функцията pipe(int pipefd[2]) в ОС GNU/Linux.**
   1. **Кои процеси не могат да използват тръбата?**
   2. **Опишете друг метод за изграждане на комуникационен канал, който дава възможност на произволни процеси да изградят и ползват канала. Допълнително искаме новоизградения канал да е достъпен само за процесите, които са го създали.**

а) Само родствени на P процеси могат да използват създадения от него pipe.

б) За комуникация между произволни процеси може да се използва именувана тръба (FIFO), но тогава тя ще бъде достъпна от всички процеси в системата. Вместо това може да се използва конекция (connection) чрез сокети.

1. **Каква задача решава инструментът spinlock (активно изчакване)? Опишете хардуерните инструменти, необходими за реализацията на spinlock. В кои ситуации не бива да се ползва spinlock?**

Spinlock се ползва за достъп до споделени данни в режим на взаимно изключване. Така се предотвратява разрушаването им, когато паралелно работещи процеси временно нарушат структурата им и създават условия за race condition.

Реализацията на spinlock използва байт(бит) lock, разположен в споделената памет. Стойността на lock показва дали паметта се ползва (lock=1) или е свободна (lock=0).

Преди изпълнението на критичната секция макросът spin\_lock(lock) проверява и сменя с 1 стойността на lock в цикъл, докато завари стойност 0. Това става със специална атомарна инструкция, примерно test\_and\_set.

След изпълнението на критичната секция макросът spin\_lock(lock) сменя с 0 стойността на lock, така освобождава споделената памет.

Преди заемането на ресурса се забраняват прекъсванията, а след освобождаването се разрешават, с цел по-бързо изпълнение на критичната секция и защита от второ влизане и spin\_lock(lock).

Операциите по манипулиране на прекъсванията, както и test\_and\_set са специфични хардуерни инструменти, без които не може да се направи удобна и бърза реализация на spinlock.

Spinlock не бива да се използва при защита на критична секция, която може да продължи дълго време или да извика отново spin\_lock(lock).