Тема 1

1. Опишете как ОС разделят ресурсите на изчислителната система, дайте примери за основните типове разделяне:

   Разделяне на пространството (памети).

   Разделяне на времето (процесори, други у-ва).

- При разпределянето на процесорното време голяма роля играе типа на процеса.
  - Има няколко вида процеси:
    - foreground/IO процеси – това са процеси, които извършват много входно-изходни операции (например чакат потребителя да въведе някакви 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 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 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 реда 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 първия процесор, който е пожелал процесорно време, е и първия на който се предоставя. Сигурно е, че няма да настъпи гладуване, т.е. всеки процес след определено време ще бъде предоставен процесор. Проблемът е, че не се взима предвид типът на процесите. Ако даден foreground 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. Например на всеки от процесите може да бъде присвоен приоритет 1000, като при всяко изпълнение на процеса, от този приоритет се изважда времето, за което е работил процеса, като се взима процесът с най-висок приоритет. Така, ако даден CPU процес е чакал прекалено дълго, със сигурност ще му бъде дадено процесорно време в определен момент. Друг вариант са fixed и floating приоритети. Всеки път, когато на процесор не му бъде дадено процесорно време се увеличава fixed приоритетът и накрая 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; по този начин не се пази максимална големина на файла, и нарастването му няма да доведе до проблеми.

2. 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. 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.
fork() - Създава нов процес (дете със стойност 0), копирайки извикващия процес(подител стойност >0).
exec() - заменят изпълнимия файл на процеса, който се изпълнява в момента, с друг изпълним файл, подаден като първи аргумент.
wait() - Спира изпълнението на извикващия процес, докато някой от процесите-дете не прекрати своето изпълнение.
waitpid() - Спира изпълнението на извикващия процес, докато процесът-дете с идентификатор(pid), равен на подадения аргумент, не промени състоянието си.

Тема 2

1. Опишете разликата между времеделене и многозадачност.

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. Следователно, многозадачността стои в основата времеделенето.

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.

2. Опишете с по едно-две изречения работата на следните системни извиквания в стандарта POSIX:

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.

Тема 3

1. 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.
Времеделението 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.

Опишете разликата между многопотребителска и многозадачна работа. Какви качества на ОС характеризират тези две понятия?

Разликата между многопотребителска и многозадачна операционна система е, че при многозадачната ОС е възможно да имаме един потребител, който да иска едновременно да изпълнява няколко задачи. При многопотребителската система имаме множество от потребители, всеки от които иска да 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.

2. Опишете с по едно-две изречения работата на следните системни извиквания в стандарта POSIX: Повтаря се с писани горе но да го има

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.
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.
pipe() - Създава неименувана тръба. Приема масив от две цели числа int fd[2], в който се съхраняват двата нови файлови дескриптора. 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.

Тема 4

1. Опишете ситуацията съревнование за ресурси (race condition), дайте пример.

Race Condition (борба или състезание за ресурси) възниква когато два или повече процеса имат достъп до споделени ресурси и се опитват да ги променят едновременно. Алгоритъмът за даване на процесорно време за изпълнение на процес е непредвидим. Ние не знаем реда, в който процесите ще се опитат да получат достъп до споделените данни и следователно резултатът от промяната в данните зависи от алгоритъма за планиране на процесите. По този начин два или повече процеса се „състезават “за достъп до данните. Този достъп до данни, често е свързан и с някаква промяна и това може да доведе до нарушаването им, като в някой момент структурата от данни на общата памет е била нарушена временно от методите ѝ. Но именно ако през това време друг метод използва структурата, то ресурса ще бъде използван докато се е актуализирал – моментите, в които се актуализира се наричат критични секции. Проблемът често възниква когато един процес изпълнява операцията „провери и след това действай “, а друг процес направи нещо със стойността на между „провери “и „действай “на първоначално споменатия процес:

```
If (x==5) {
Y=x*5
}
```

Въпросът е, че може и да е 25, но може и да бъде всичко в зависимост от това дали друг процес е бъркал в критичен момент и в критична секция на кода. Няма как да знаем дали това се е случило. За да предотвратим появата на борба за ресурси, обикновено поставяме заключване около споделените данни, за да гарантираме, че само един процес може да има достъп до тях за определено време. Тоест нещо подобно:

```
Lock(x)
If (x==5) {
Y=x*5
}
Unlock(x)
```

Опишете накратко инструментите за избягване на race condition:

(а) 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: Инструкции, които променят бит и го прочитат след това като една единна непрекъсната инструкция.

(б) 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.

(в) 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.

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;

/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
/usr/lib - обектни файлове и библиотеки
/var/log - директорията, съдържаща всички log (журнални) файлове

Тема 5

1. Хардуерни инструменти за защита (lock) на ресурс:

(a) 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.

(b) test and set

test and set: Тази инструкция се използва, за да се пише в паметта и да се върне старата стойност като атомарна операция.

(c) 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.

2. 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 1 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 със race condition - т.нар spinlock. При него имаме един допълнителен бит - lock, който пази стойност 0 ако ресурса 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 този инструмент 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 процесора 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.

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;

/var – системни файлове - logging files, mail directories, printer spool, etc.; за данни, не на потребителя, а за споделени данни
/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

/home – home директориите на потребителите в системата

/usr/doc - съдържа всичката необходима документация.

Тема 6

1. Опишете понятията приспиване и събуждане на процес (block/wakeup).

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 го събужда (wakeup).

Семафор – дефиниция и реализация.

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.

```
struct Semaphore {

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

        constructor init(int _cnt) {
                cnt = _cnt
        }

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

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

Опишете разликата между слаб и силен семафор.

Силен семафор е този, който използва обикновена опашка за множеството. Слаб семафор е този, при който има някакъв критерии, който се ползва за приоритет при обслужването на процес. Слабостта идва от факта, че такъв критерии може да причини така нареченото „гладуване“ на процеси/нишки с най-нисък приоритет, което се счита за проблем със синхронизацията. Не искаме да имаме процеси, които да чакат твърде много.

2. Опишете накратко различните видове специални файлове в Linux:

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

сокети - 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.

Тема 7

1. 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 си дефинираме семафор s така и добавим следните синхронизационни инструкции в кода на P:

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.

2. Качества и свойства на конкретните файловите системи, реализирани върху block devices. Ефективна реализация

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
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

отлагане на записа

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. 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.

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.

Тема 8

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

pipe с буфер – тръба, съхраняваща n пакета информация. Използване на семафорите като броячи на свободни ресурси.

Тръбата се използва от няколко паралелно работеши „изпращачи/получатели“ на байтове. Процесите изпращачи – слагат байтове в края на опашката, а получателите – четат байтове от началото на опашката

Опашка (или масив) Q с елемента - В началото тази опашка ще 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

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)	
READ:	**WRITE:**
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() …

2. Права и роли в UNIX, команда chmod

Права – u/g/o user/group/others

Роли – r/w/x read/write/execute

- 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.
    - 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.
- Роли – r/w/x read/write/execute
  - Всеки файл и директория има 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.
  - Запис: Дава правомощието да променяме съдържанието на файл. При директориите, дава правото да добавяме, премахваме и преименуваме файлове, съхранявани в директорията.
  - Изпълнение: Ако разрешението за изпълнение не е зададено, не можем да стартираме изпълним файл. При директориите ситуацията е малко по-особена. 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.

1. 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.

4 правила, по които може да установим, че има deadlock:

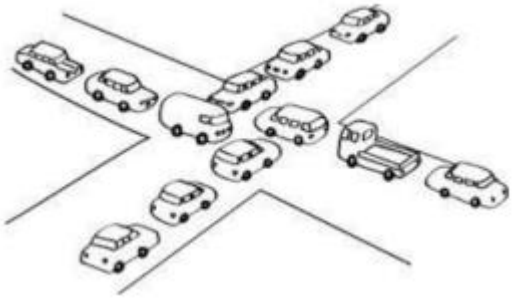
Поне един процес да използва споделен ресурс монополно, тоест да има ексклузивни права над ресурса

Може да се случат поредици, при които процес взима даден ресурс и чака да вземе следващия.

Това ползване на споделен ресурс в 1. да не може да бъде прекъсвано от външен фактор.

Усовие за кръгово (циклично) изчакване (както при синхронизационния проблем с философите и спагетите.

Пример:



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.

Starvation настъпва, когато „алчните“ нишки използват споделените ресурси прекалено дълго и ги правят недостъпни за останалите процеси за дълги периоди. 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.

Пример: задача за философите и макароните от другия файл че е много красиво

2. 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), като когато добавим някакво външно устройство (напр. когато свържем флашка към компютъра), това устройство се появява във файловата система. Към всеки файл, освен името и съдържанието, се асоциират и други атрибути - тип на файла ("-

" - обикновени файлове, "d" - директории, "c" - символно устройство/character special device (напр. клавиатурата), "b" - блоково устройство/block special device – външно у-во, което съхранява масив от байтове, "p" - именувана тръба (FIFO), "s" - socket/конектор – крайна точка за комуникация), права за достъп, потребител, 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.

Файлове и директории команди –

cd - променят текущата директория

mkdir – създава директория

rmdir - трие празна директория

cp – копира файл/ове на посочено място

mv – мести файл на посочено място, преименува файл

rm – трие файл/ове

Тема 10

1. Процеси в многозадачната система.

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).

Превключване, управлявано от синхронизация.

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.

Превключване в система с времеделене – timer interrupt.

Работещ процес може да промени състоянието си на блокиран, 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 и 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 процес, преминава в работещ, 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. Когато спящият процес е приспан заради очакване на момент от времето, то той може да стане активен, като този процес се инициира от timer.

2. Опишете функционалността на следните команди в Linux:

ls – показва съдържанието на директорията
who – показва активните в момента потребители
find – търси файлове/директории по зададени параметри
ps – подробна информаиця за активните процеси в момента на извикване
top – като ps, подробна информация за процсите в реално време

Тема 11

1. Възможни състояния на процес. Механизми и структури за приспиване/събуждане.

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:

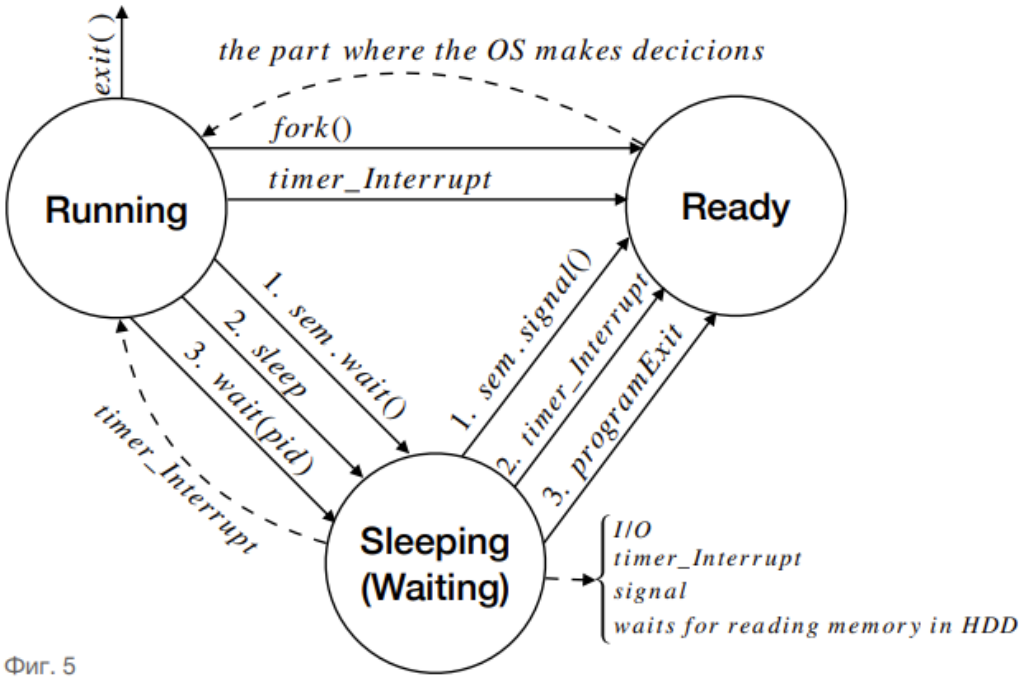
- Running – изчисляват се в момента;
- Ready – в момента няма процесор за тях, но при следващия такт те ще получат управление.

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)

Диаграма на състоянията и преходите между тях.



Фиг. 5

2. Опишете функционалността на следните команди в Linux:

vi – отваря на конзолата текстов редактор

tar – създава архива

gcc – компилира програми на C и C++

Тема 12

1. 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;

Ранните механизми за защита на ползваната памет:

Разделяне на паметта на интервали-най-прост механизъм; Разглеждаме паметта като един непрекъсна блок, то 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-прост такъв механизъм е да има специални регистри, в които се указва къде е този интервал; в ранните машини интервала пряко се показва(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)

Друг механизъм е регистрите да се обозначават Base и да се знае дължината на адресното пространство Size; като 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

Суматори с ускорен пренос- в по-модерните системи

В по-ранните системи(mainframe) 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: защитен интервал(base,size); 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

Чрез виртуална машина(не изисква хардуер)

Ако реалната памет я разглеждаме като един непрекъснат блок, да отделим един интервал за всеки процес. Когато на процесът му се наложи да комуникира с друг процес, да се обърне към ядрото и то да изпълни задачата.
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.

Йерархия на паметите – кеш, RAM, swap.

Паметта е 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 - по нетрайни. При междинният клас памети(динамични RAM чипове) – те са евтини, относително бързи и събират много информация, но ако не биват често обновявани, паметта се губи, защото се губи сигнала поради факта,че много малко електроника се влага в рам (енергозаивисми). Има различни видове памети в хардуера: Качества на паметите(енергозависимост, скорост на работа, брой презаписи)
Кеш-скъпи, но бързи; за пресмятане да подходящи; поддържа данните на паметта; кеш е спомагателна памет за ускоряване обмена на данни между различните нива в йерархията на паметта. 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. Може да постигне различни степени на ускорение в зависимост от вида на обменяните данни, от алгоритъма за избор на данните за копиране и от съвместимостта помежду им. За да са по ефективни и по-ефикасни в употребата на данни, кешовете са сравнително малки
RAM- по-голямата част от процеса е тук; това е паметта, която процесора директно ползва. В нея се разполагат стека, кода на програмата, данните(така бързо става четенето и писането на байтове)
Хард диска-вечен, но те са механични устройства => тежък, хабят енергия; Флаш паметта-има деструктивност на паметта;броят презаписи е ограничен
Swap- когато части от процеса са вкарани в по-бавни хардуерни ресурси; Това се нарича swapping-временно съхранение на рядко използваните процеси. Swap-памет върху диска, която е част от Ram-а
Кеш на централния процесор (CPU кеш), е кеш използвана от централният процесор (CPU) на компютъра, за да ускори времето за достъп до информация на RAM паметта. Повечето коммпютърни процесори имат няколко независими една от друга кеш памети. Всички инструкции и адреси за запазване, които се ползват от процесора трябва да идват от RAM. Въпреки че RAM е много бърза, все пак значително време е нужно за ЦП да я достъпи. RAM паметта е много по-голяма от кеш паметта. Колкото е по-бърза паметта,толкова по-скъпа става тя. Swap-ването е 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 си продължи изпълнението. Резервният компонент попринцип е хард диск, който има бърз достъп и е достатъчно голям, за да си набави копия на всички изображения на паметта за потребителите.

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). Например, при Windows ако няма достатъчно оперативна памет за изпълнение на програма или операция, операционната система използва виртуална памет. Тя се разделя на различни статични парчета:

(1) Статична част (read only) от кода-частта която го описва, тя се изчислява колко е дълга при създаване на процеса, използва се и от други процеси(споделена област-така се пести памет):

1. Има блок с фиксирани размери, където се разполага кода на програмата, памет, която няма да се променя,записана е самата програма, изчислява се размера предварително при стартирането;
2. Има парчета за споделени библиотеки, които са достъпни едновременно за няколко процеса и се предоставят на процеса;

- Динамична част(read-write памет)

1. Стекове-намира към края на адресното пространство; Нужен на процеса е за управление на програмата, в него съществува реализацията на структурите за управление на програмата предоставени от езика за програмиране(функции, променливи, подпрограми). Стекът е проста структура, заема обем от паметта, който може да нараства и да намалява.
2. 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)
3. Блок за статична памет-променливи, които при стартиране на процеса се заделя винаги тази памет; място за данни-обем на масиви, константи и др.

- Структурата на виртуалната памет може да се разгледа по следния начин:

1. Данни-Глобални променливи, константи, статични променливи от програмата
2. Kernel - Докато това е част от адресното пространство, то не може да се адресира директно от някой процес. Трябва да се адресира чрез системни извиквания.
3. Стек - Използва се от програмата за променливи и запазване. Расте и и си смалява размера в зависимост от това какви практики са извикани и какво са техните изисквания за размера на стека (8MB в размер).
4. Heap - Използва се за някои видове работни хранилища.
5. Споделени библиотеки - Споделените библиотеки са позиционно независими и така те могат да бъдат споделени от всички програми, които искат да ги използват.

2. Опишете функционалността на следните команди в shell:

echo – извежда на стандартния изход текст подаден като аргумент

read – прочита ред подаден на стандатния вход

test – проверява типове на файлов и сравнява стойности

if – изпълнява команди ако условието е изпълнено if [условие];then команди fi; elif – допълнително условие; else – ако не е изпълнено условието от if се изпълняват командите от else

for – for елемент in множество; do команди done за всеки елемент от множеството изпълнява командите

while – while условие; do команди done изпълнява циклично докато условието е вярно.

Тема 13

1. 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). Инструмента, който използваме за оптимизиране на работата на MMU, се нарича TLB (Translation Lookaside Buffer). В зависимост от схемата за транслация, TLB може да се реализира по различни начини За конкретния случай.

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 в физически адреси, докато записите от таблица със сегменти – за преобразуване на виртуалните адреси в сегментни адреси.

2. файлови дескриптори, номера на стандартните fd, пренасочване

| - прави композиция на две програми; p1 | p2 изходните данни от първата се подават като входни на втората
Първоначално, shell-ът казва на ядрото да създаде pipe (тръба). След това шелът(shell) започва да се размножава (fork()). И двата новообразувани процеса изпълняват една и съща програма, но всеки от тях си работи в собствена памет. Новият процес(shell1) наследява всичко от стария. Старият процес наричаме родител. Той остава да работи там, където е бил, докато новият работи с програмата на стария, но си прави копие на цялата памет (прави си нов стек и т.н.).
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.

cat – извежда съдържанието на файл/ове на стандартния изход

grep – търси редове отговарящи на образец и ги извежда

cut – извежда само избрани колони отделени по зададен разделител

sort – сортита редове и ги извежда

wc – брои редове, думи, символи и байтове

tr – заменя символ/и с други символи (-d трие символи)

1. Избройте видове събития, причиняващи повреда на данните във файловите системи.

   Опишете накратко стандарта RAID5.

   Какво е журнална файлова система? Това го има 100 пъти писна ми да го пиша

2. Свързване и допускане до UNIX система – login.

   Програмата за вход се използва за установяване на нова сесия със системата. Обикновено се извиква автоматично чрез отговор на подканата "login:" на терминала на потребителя. login може да е специален за обвивката и може да не се извиква като подпроцес. Когато се извиква от шел, входът трябва да се изпълни като exec login, което ще накара потребителя да излезе от текущата обвивка (и по този начин ще попречи на новия влезнал потребител да се върне към сесията на повикващия). Опит за изпълнение на влизане от която и да е обвивка, но обвивката за влизане ще доведе до съобщение за грешка.

   Конзола – стандартен вход, стандартен изход, стандартна грешка.

   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.

   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

   Тема 15

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

   При асинхронна операция, процесът не се приспива, но това се заплаща с цената на върнат на части резултат от операцията и това се индикира със специален код за грешкаИзползването на

такъв тип операции позволява на един процес да извършва паралелна комуникация по няколко канала с различни устройства или процеси, без да бъде блокиран.

Класически пример са графинчните приложения (използващи информация от мишка, клавиатура, touch screen и т.н.), друг пример е WEB-browser, той трябва да реагира на входни данни от клавиатура и мишка, както и на данните, постъпващи от интернет.

2. Опишете с по едно-две изречения работата на следните системни извиквания в стандарта POSIX:

socket() – Използва се за създаване на комуникационен канал от тип „конекция“. Създава сокет и връща файловия дескриптор, асоцииран с него.

bind() – “именуване на сокет”, Присвоява адрес на сокет, създаден със socket()

connect() – запитване от клиента към socketa дали може да установи връзка

listen() – Означава сокета, подаден като аргумент, като готов да приема нови входящи заявки за свързване.

accept() – приемането на запитването от клиента от connect() от страна на именувания socket

Тема 16

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

В съвременната операционна система има 2 слоя на абстракция:
1. Пространство на имената – всички дълготраини обекти, които съществуват в системата. Това е абстрактната файлова система;
2. Реализацията – как са представени тези обекти и как са съпоставени на хардуера (всеки информационен обект, как и къде ще бъде в хардуера). В UNIX, абстрактната файлова система е едно кореново дърво.

- "-" – обикновени файлове;
- "d" – папка или директория;
- "l" – символен линк (друго име на файл – съществуващ или не);
- "c" – символно устройство (character device);
- "b" – масив от баитове, който не е RAM паметта на компютъра, нито процесора и неговите регистри, а е някакво външно устройство, което съхранява някакъв масив от байтове и може да се ползват или записват байтове където решим (може да се индексира)
- "p" – именувана тръба (FIFO);
- "s" – крайна точка за комуникация (socket). Процес сървър, който предоставя връзки за съответни услуги (служи за да може клиента да намира сървъра).

2. Една от класическите задачи за синхронизация се нарича „Задача за читателите и писателите“ (readers-writers problem).

   Опишете условието на задачата и решение, използващо семафори.

- 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()

Тема 17

1. 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 атрибути – права за достъп, информация за собственост, група, размер на файла, дата на създаване, дата на последна промяна, дата на достъпване.

Опишете накратко реализацията и целта на следните инструменти:

(а) отлагане на записа, 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.

(б) поддържане на буфери (кеширане) на файловата система.

2. Опишете как се изгражда комуникационен канал (connection) между процес-сървер и процес-клиент със следните системни извиквания в стандарта POSIX: има ги в предия въпрос

socket() – Използва се за създаване на комуникационен канал от тип „конекция“. Създава сокет и връща файловия дескриптор, асоцииран с него.
bind() – “именуване на сокет”, Присвоява адрес на сокет, създаден със socket()
connect() – запитване от клиента към socketa дали може да установи връзка
listen() – Означава сокета, подаден като аргумент, като готов да приема нови входящи заявки за свързване.
accept() – приемането на запитването от клиента от connect() от страна на именувания socket

Тема 18

1. Опишете накратко основните комуникационни канали в ОС Linux.

Кои канали използват пространството на имената и кои не го правят?

Неименувана тръба (pipe) се създава чрез системното извикване pipe(fd[2]). То връща два файлови дескриптора на мястото на елементите на подадения масив int fd[2]. fd[0] съхранява файловия дескриптор за четене, а fd[1] съхранява файловия дескриптор за писане. Тъй като тази тръба не е именувана, тя е видима само за процеса, който я е създал, както и от наследниците му (децата му, децата на децата му и т.н.). Тя не използва пространството на имената.

Именувана тръба (FIFO) – този вид тръба се създава чрез библиотечното извикване mkfifo(...). За разлика от неименуваната, този вид тръба е видима за всички процеси в системата и може да бъде ползвана от тях. Тя се използва за осъществяване на комуникация между два процеса, които не са задължително в роднинска връзка. Този вид тръба се обвързва с име, откъдето следва, че тя използва пространството на имената.

Писане във файл и четене от файл. Чрез системното извикване open(filepath, open_flags, ...) се създава файлов дескриптор, който сочи към единя край на комуникационния канал, който се изгражда в ядрото на операционната система и е за писане, а на другия край сочи към файла. Чрез системните команди read(int fd, void *buf, size_t cnt) и write(int fd, const void *buf, size_t cnt) може да се чете и пише от и във файла. Този вид комуникационен канан използва пространството на имената.

Конекция (socekt). Това е именуван обект, който играе ролята на крайна точка за комуникация между отдалечени обекти и който се използва за адрес при изграждането на връзка с друг процес. Тук има два вида процеси – единият се нарича сървър и работи като такъв (ще обслужва други процеси), а другият – клиент. Предимствата на сокетите пред тръбите е, че процесите, с които може да се изгради връзка, могат да не се намират в една и съща система.

2. Опишете какви изисквания удовлетворява съвременна файлова система, реализирана върху блочно устройство (block device).

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
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

Опишете предназначението на журнала на файловата система.

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. 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.