Тема 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).
- ✓ *Опишете с по едно-две изречения работата на следните системни извиквания в стандарта POSIX:*
  pipe() dup2() fork() exec() wait() waitpid()

ОС разделят ресурсите на изчислителната система с цел ефективно и справедливо използване на наличните хардуерни и софтуерни компоненти от множество процеси и потребители.
Основните типове разделяне са:

1. **Разделяне на пространството:**
   Тук ресурсите, като оперативната памет (RAM) и дисковото пространство, се разпределят логически между различните процеси или потребители. Пример за това е виртуалната памет – всеки процес работи в собствено виртуално адресно пространство, което чрез техники като страниране или сегментиране се съпоставя с физическата памет. Това позволява на процесите да използват по-голям адресен обхват, отколкото е наличният физически размер на RAM.

2. **Разделяне на времето:**
   При този подход ресурсите, като процесорното време и достъпа до други устройства (например принтери или мрежови адаптери), се разпределят на малки времеви интервали – кванти. Всеки процес получава редовен дял от процесорното време, така че да изглежда, че работи едновременно с другите, въпреки че физически процесорът изпълнява само една задача в даден момент. Този метод осигурява ефективно използване на процесора и намалява времето за реакция на системата.

pipe()
Pipe() създава анонимен комуникационен канал с два края – един за четене и един за писане – чрез който може да се прехвърлят байтове между процесите. Аргументът представлява масив от два файлови дескриптора, като процесът, който създава канала, получава достъп до двата края.

dup2()
Dup2() дублира файлов дескриптор, като копира съдържанието на един дескриптор в друг, позволявайки пренасочване на входа или изхода. Ако целевият дескриптор е вече отворен, той се затваря преди дублирането, което гарантира, че новият дескриптор сочи към желания ресурс.

fork()
Fork е единственият метод в UNIX за създаване на нов процес. При извикване на fork(), текущият процес (родител) копира цялото си адресно пространство (локална памет, защитени данни и файлови дескриптори) в нов процес (дете), като двете програми работят паралелно. Разликата се определя от стойността, върната от fork():

< 0 – грешка;

= 0 – кодът, изпълняван от детето;

> 0 – идентификатор (PID) на детето, върнат на родителя.

exec()
Exec() е семейство от системни извиквания, които заменят текущия процес с нова програма. При успешно изпълнение на exec(), текущото съдържание на паметта се изтрива и новият изпълним файл се зарежда, така че процесът продължава като напълно нова програма.

wait()
Wait() позволява родителският процес да изчака завършването на някой от своите наследници. Ако нито един наследник не приключи, родителят се блокира, докато не бъде завършен някой процес.

waitpid()
Waitpid() е по-гъвкав вариант на wait(), който позволява на родителя да изчака завършването на конкретен процес по негов PID или да използва опции за не блокиращо изчакване. Това осигурява по-фин контрол върху синхронизацията с детските процеси.

Тема 2

- ✓ *Опишете разликата между времеделене и многозадачност.*
- *Какви ресурси разделя еднозадачна, еднопотребителска ОС?*
- *Опишете с по едно-две изречения работата на следните системни извиквания в стандарта POSIX:*
  open() close() read() write() lseek()

Времеделенето и **многозадачността** са техники за абстракция, използвани за ефективно разпределение на системните ресурси, като например оперативната памет (RAM).
Многозадачността представлява по-прост механизъм, приложим още в първите операционни системи. При нея всеки процес получава изчислително време последователно, като превключването между процесите може да става в сравнително дълги интервали. Това означава, че не може да се установи в момента кой процес работи – процесите „работят тайно“, а важното е фиктивното използване на изчислителната система.
При **времеделенето** се цели няколко активни процеса да изглеждат, че работят едновременно, като всеки процес има впечатление, че разполага с целия процесор. Това е по-сложна техника за разделяне, при която времето се разделя на малки кванти, които след това се предоставят последователно на активните процеси.

Тема 3

- ✓ *Дайте кратко определение за: многозадачна ОС, многопотребителска ОС, времеделене.*
- *Опишете разликата между многопотребителска и многозадачна работа.*
- *Какви качества на ОС характеризират тези две понятия?*

- *Опишете с по едно-две изречения работата на следните системни извиквания в стандарта POSIX:*
  open() close() lseek() pipe() dup2()

Многозадачна ОС: Операционната система, която може да изпълнява няколко задачи (процеса) едновременно чрез превключване на процесорното време между тях.

Многопотребителска ОС: Операционна система, която позволява на множество потребители да използват системните ресурси едновременно, като гарантира изолация и контрол на достъпа до ресурсите.

Времеделене: Техника за разпределяне на процесорно време, при която времето се разделя на малки интервали (кванти) и всеки активен процес получава свой дял, така че да изглежда, че работи самостоятелно.

Тема 4

- ✓ *Опишете ситуацията съревнование за ресурси (race condition), дайте пример.*
- ✓ *Опишете накратко инструментите за избягване на race condition:*
  (а) дефинирайте критична секция, атомарна обработка на race conditio.
  (б) инструменти от ниско ниво, специфични хардуерни средства.
  (в) инструменти от високо ниво, които блокират и събуждат процес.
- ✓ *Каква е спецификата на файловете в следните директории в Linux:*
  /etc /dev /var /boot /usr/bin /home /usr/lib /var/log

Състезанието за ресурси (race condition) възниква, когато два или повече процеси (или нишки) едновременно достъпват споделен ресурс – например структура от данни в общата памет – и поне един от тях я модифицира без подходяща синхронизация. Това може да доведе до нарушаване на инвариантите (състояния, които трябва да останат непроменени) на структурата, като поради това може да се загубят или повредят данни.

Пример:
Да предположим, че имаме структура от данни, съдържаща брояч, който винаги трябва да бъде положително число. Ако два процеса едновременно се опитат да увеличат стойността на брояча без да използват механизъм за синхронизация, е възможно и двамата да прочетат една и съща стара стойност, да изчислят новата стойност независимо един от друг и след това да запишат резултата. В този случай едно от увеличенията може да бъде загубено, защото промените се базират на една и съща начална стойност.

За да се предотврати race condition, се дефинира **критична секция** – това е част от кода, където се извършват операции, които временно нарушават нормалното (статично) състояние на структурата. Докато един процес работи в критичната секция, останалите процеси трябва да бъдат блокирани, за да не се опитват да модифицират същия инвариант. Атомарната операция представлява неделима операция, която се изпълнява изцяло или изобщо не се изпълнява – тя гарантира, че промените се извършват без да се прекъсват от други процеси, като по този начин се избягва race condition

Spinlock(инструемнт от ниско ниво):
В съвременните паралелно работещи ядрови системи методът с забрана на

прекъсванията не е приложим. Вместо това се използва допълнителен бит – наречен lock – който показва дали данните в критичната секция са свободни или се използват. При този подход процесът, който иска достъп до критичната секция, извиква процедурата spin_lock, която чрез операцията test-and-set проверява и "заключва" ресурса. След приключване на критичната секция ресурсът се освобождава чрез spin_unlock.

Семафор(инструменти от високо ниво)
Семафорът е синхронизационен механизъм от високо ниво, дефиниран от Дейкстра около 1965 г. Той се използва за защита на информационен ресурс от прекомерна употреба, като контролира броя на процесите, които могат да имат достъп до даден ресурс едновременно, чрез смяна на *контекста.

* *Смяната на контекста (context switch) представлява процес, при който операционната система прекъсва изпълнението на текущия процес, запазва неговото състояние (регистрите, програмния брояч, стековете и т.н.) и зарежда състоянието на друг процес, който трябва да бъде изпълнен. Това позволява на един процесор да обслужва множество процеси, като всеки получава дял от процесорното време. Този процес е от съществено значение за постигането на многозадачност, но носи и известно натоварване, тъй като смяната на контекста изисква допълнителни изчислителни ресурси за запазване и възстановяване на състоянията на процесите.*

Основни компоненти:

- **cnt:** Това е броячът, който определя колко процеса могат едновременно да влязат в критичната секция. Когато cnt достигне 0, процесите, които се опитват да влязат, се блокират. Ако cnt стане отрицателен, неговата стойност показва броя на процесите, чакащи достъп.
- **L:** Множество (или опашка) от процеси, които са блокирани и изчакват освобождаване на ресурса. При силния семафор L представлява обикновена опашка, като се събужда първият влязъл процес.
- **spinlock:** Малък механизъм (например няколко байта), който се използва за осигуряване на взаимно изключване при промяна на променливите cnt и L. Чрез него се реализират атомарни операции като test-and-set.

Силен семафор е синхронизационен механизъм, който гарантира честен и детерминиран достъп до споделени ресурси, като управлява блокираните процеси чрез опашка (обикновено по принципа "първият влязъл – първият излязъл" – FIFO). Това означава, че когато даден ресурс се освобождава чрез операцията signal(), първият процес, който е блокиран и е изчаквал достъп, се събужда. Така се предотвратява неравномерно разпределение на ресурсите и се избягва възможността някои процеси да бъдат "гладни" (starvation), докато други получават прекомерен достъп.

Реализация:

Инициализация (s.init(cnt_0)):
При създаването на семафора задаваме началната стойност cnt_0, която определя колко процеса могат да използват ресурса едновременно.

init (cnt_0):
  cnt = cnt_0
  L= ∅

Операция s.wait() – заявка за влизане в ресурса:
Процесът, който иска да влезе в критичната секция, извиква s.wait(). При това:

- Влизаме в критична секция чрез spinlock.
- Намаляваме брояча cnt с 1.
- Ако cnt става по-малък от 0, това означава, че ресурсът е зает и текущият процес се добавя към опашката L и се блокира.
- Ако cnt остава 0 или по-голямо, процесът продължава изпълнението си.

```
s.wait():
  spin.lock(lock)
  cnt = cnt – 1
  if cnt < 0 then:
     L.put(pid)      // Добавяме идентификатора на процеса в опашката
     spin.lock(lock)   // Задържаме spinlock-а, докато процесът бъде събуден
     block()         // Блокираме процеса
  else:
     spin.unlock(lock)
```

Операция s.signal() – излизане и освобождаване на ресурса:
Когато процесът приключи с използването на ресурса, извиква s.signal(), като:

- Влизаме в критичната секция чрез spinlock.
- Увеличаваме брояча cnt с 1.
- Ако cnt остава по-малко или равно на 0, това означава, че има блокирани процеси в опашката L. Изваждаме първия процес от L, освобождаваме spinlock-а и го събуждаме.
- Ако cnt става по-голям от 0, просто освобождаваме spinlock-а.

```
s.signal():
   spin.lock(lock)
   cnt = cnt + 1
   if cnt <= 0 then:
      pid = L.get()    // Изваждаме процес от опашката
      spin.unlock(lock)
      wakeup(pid)     // Събуждаме процеса
   else:
      spin.unlock(lock)
```

/etc – Тази директория съдържа системни конфигурационни файлове и настройки, специфични за дадената инсталация на операционната система, като информация за потребителите, мрежовите настройки, стартиращите услуги и други системни параметри.

/dev – В тази директория се намират файловете, които представляват интерфейси към физически или виртуални устройства (например дискове, терминали, принтери), позволявайки на системата да комуникира с тях чрез стандартни системни извиквания.

/var – Тук се съхраняват променливи данни, общи за потребителите , които се обновяват по време на работа на системата, като лог файлове, съобщения за

грешка, бази данни, временни файлове, необходими както за системните, така и за потребителските приложения.

/boot - Съдържа файлове, необходими за стартиране на системата, включително ядрото.

/bin - *Тази директория съдържа основни изпълними файлове и команди, които са критични за функционирането на системата, дори когато останалите файлови системи не са монтирани (например при стартиране или аварийно възстановяване).*

/usr/bin - Тази директория включва по-широк набор от потребителски команди и приложения, които не са критични за базовата система, но осигуряват допълнителни функционалности за ежедневната употреба на потребителите **Разликата между тях е, че /bin съдържа минималния набор от команди, нужни за основните операции на системата, докато /usr/bin съхранява допълнителни програми, използвани за по-разнообразни задачи и приложения.*

/home - Тук се намират личните директории на потребителите, където всеки има отделна поддиректория за съхранение на своите файлове, настройки и лични данни.

/usr – Тук са програмите и файловете, които се използват от конкретен потебител

/usr/lib - Съдържа споделени библиотеки и модули, използвани от изпълними файлове в /usr/bin и от други програми.

/var/log - Съдържа лог файлове, в които системата записва съобщения, грешки и други събития, свързани с работата на услугите.

Тема 5

- *Хардуерни инструменти за защита (lock) на ресурс:*
- *(a) enable/disable interrupt*
- *(b) test and set*
- *(c) atomic swap*
- ✓ *Опишете инструмента spinlock, неговите предимства и недостатъци.*
- ✓ *Каква е спецификата на файловете в следните директории в Linux: /etc /dev /var /proc /bin /home /usr/doc*

Метод чрез забрана на прекъсванията:
При този подход в началото на критичната секция се забраняват всички прекъсвания, за да се осигури, че само текущият процес има достъп до ресурсите. В края на критичната секция прекъсванията се разрешават отново. Този метод беше типичен за първите еднопроцесорни операционни системи.

Spinlock (за многоядрени системи):
В съвременните паралелно работещи ядрови системи методът с забрана на прекъсванията не е приложим. Вместо това се използва допълнителен бит – наречен lock – който показва дали данните в критичната секция са свободни или се използват. При този подход процесът, който иска достъп до критичната секция, извиква

процедурата spin_lock, която чрез операцията test-and-set проверява и "заключва" ресурса. След приключване на критичната секция ресурсът се освобождава чрез spin_unlock.

Пример за псевдокод:

```
spin_lock:
  disable_interrupts()     // Забраняване на прекъсванията
  R = test_and_set(lock)
  if R == 0 then goto critical_section
  enable_interrupts()      // Разрешаване на прекъсванията
  goto spin_lock
critical_section:
  // Изпълнение на критичната секция
spin_unlock:
  lock = 0
  enable_interrupts()      // Разрешаване на прекъсванията
```

Важно е този метод да не бъде рекурсивен и да се избягват прекъсвания по време на изпълнението на критичната секция, особено при драйвери, където се задават конкретни правила за управление на прекъсванията. Това е единствения механизъм от ниско ниво и е сравнително бърз. Недостатък е, че докато единия процес работи по критичната секция, другите процеси циклят, по-разумно е да бъдат спрени временно.

/etc – Тази директория съдържа системни конфигурационни файлове и настройки, специфични за дадената инсталация на операционната система, като информация за потребителите, мрежовите настройки, стартиращите услуги и други системни параметри.

/dev – В тази директория се намират файловете, които представляват интерфейси към физически или виртуални устройства (например дискове, терминали, принтери), позволявайки на системата да комуникира с тях чрез стандартни системни извиквания.

/var – Тук се съхраняват променливи данни, общи за потребителите , които се обновяват по време на работа на системата, като лог файлове (съобщения за грешки), бази данни, временни файлове, необходими както за системните, така и за потребителските приложения.

/proc – Това е виртуална файлова система, предоставяща динамична информация за състоянието на системата и текущо изпълняваните процеси, както и системни параметри, генерирани от ядрото.

/bin – Директорията /bin съдържа основни изпълними файлове и команди (например ls, cp, mv, cat), които са необходими за функционирането на системата и са достъпни за всички потребители.

/home - Тук се намират личните директории на потребителите, където всеки има отделна поддиректория за съхранение на своите файлове, настройки и лични данни.

/usr – Тук са програмите и файловете, които се използват от конкретен потебител

/usr/doc – Тази директория съдържа документацията за инсталираните програми и системни компоненти, включително ръководства, man страници и други информационни файлове, предназначени за потребителска употреба.

Тема 6

- ✓ *Опишете понятията приспиване и събуждане на процес (block/wakeup).*
- ✓ *Семафор - дефиниция и реализация.*
- ✓ *Опишете разликата между слаб и силен семафор.*
- *Опишете накратко различните видове специални файлове в Linux:*
- *външни устройства, именувани в /dev, псевдофайлове в /proc*
- *линкове - твърди и символни, команда ln*
- *сокети*

Приспиването (блокирането) на процес означава, че процесът се поставя в състояние на изчакване, когато необходимите събития (например входно-изходни операции) не са настъпили. Докато е приспан, процесът не използва процесорно време. За да се възстанови и да продължи изпълнението си, от друга част на системата се извиква операцията **wakeup**, като й се подава идентификаторът (или друг аргумент), който указва кой процес трябва да бъде събуден. Това събуждане сигнализира, че необходимото събитие е настъпило и процесът може да бъде върнат в активното състояние.

Семафор(инструменти от високо ниво)

Семафорът е синхронизационен механизъм от високо ниво, дефиниран от Дейкстра около 1965 г. Той се използва за защита на информационен ресурс от прекомерна употреба, като контролира броя на процесите, които могат да имат достъп до дадения ресурс едновременно, чрез смяна на *контекста.

* Смяната на контекста (context switch) представлява процес, при който операционната система прекъсва изпълнението на текущия процес, запазва неговото състояние (регистрите, програмния брояч, стековете и т.н.) и зарежда състоянието на друг процес, който трябва да бъде изпълнен. Това позволява на един процесор да обслужва множество процеси, като всеки получава дял от процесорното време. Този процес е от съществено значение за постигането на многозадачност, но носи и известно натоварване, тъй като смяната на контекста изисква допълнителни изчислителни ресурси за запазване и възстановяване на състоянията на процесите.

Основни компоненти:

- cnt: Това е броячът, който определя колко процеса могат едновременно да влязат в критичната секция. Когато cnt достигне 0, процесите, които се опитват да влязат, се блокират. Ако cnt стане отрицателен, неговата стойност показва броя на процесите, чакащи достъп.
- L: Множество (или опашка) от процеси, които са блокирани и изчакват освобождаване на ресурса. При силния семафор L представлява обикновена опашка, като се събужда първият влязъл процес.
- spinlock: Малък механизъм (например няколко байта), който се използва за осигуряване на взаимно изключване при промяна на променливите cnt и L. Чрез него се реализират атомарни операции като test-and-set.

Реализация:

1.Инициализация (s.init(cnt_0)):
При създаването на семафора задаваме началната стойност cnt_0, която определя колко процеса могат да използват ресурса едновременно.
init (cnt_0):
  cnt = cnt_0
  L= ∅
2.Операция s.wait() – заявка за влизане в ресурса:
Процесът, който иска да влезе в критичната секция, извиква s.wait(). При това:
- Влизаме в критична секция чрез spinlock.
- Намаляваме брояча cnt с 1.
- Ако cnt става по-малък от 0, това означава, че ресурсът е зает и текущият процес се добавя към опашката L и се блокира.
- Ако cnt остава 0 или по-голямо, процесът продължава изпълнението си.

```
s.wait():
  spin.lock(lock)
  cnt = cnt – 1
  if cnt < 0 then:
     L.put(pid)      // Добавяме идентификатора на процеса в опашката
     spin.lock(lock)   // Задържаме spinlock-a, докато процесът бъде събуден
     block()         // Блокираме процеса
  else:
     spin.unlock(lock)
```

3.Операция s.signal() – излизане и освобождаване на ресурса:
Когато процесът приключи с използването на ресурса, извиква s.signal(), като:
- Влизаме в критичната секция чрез spinlock.
- Увеличаваме брояча cnt с 1.
- Ако cnt остава по-малко или равно на 0, това означава, че има блокирани процеси в опашката L. Изваждаме първия процес от L, освобождаваме spinlock-a и го събуждаме.
- Ако cnt става по-голям от 0, просто освобождаваме spinlock-a.

```
s.signal():
   spin.lock(lock)
   cnt = cnt + 1
   if cnt <= 0 then:
     pid = L.get()   // Изваждаме процес от опашката
```

```
    spin.unlock(lock)
    wakeup(pid)     // Събуждаме процеса
  else:
    spin.unlock(lock)
```

Силен семафор е синхронизационен механизъм, който гарантира честен и детерминиран достъп до споделени ресурси, като управлява блокираните процеси чрез опашка (обикновено по принципа "първият влязъл – първият излязъл" – FIFO). Това означава, че когато даден ресурс се освобождава чрез операцията signal(), първият процес, който е блокиран и е изчаквал достъп, се събужда. Така се предотвратява неравномерно разпределение на ресурсите и се избягва възможността някои процеси да бъдат "гладни" (starvation), докато други получават прекомерен достъп.

Слаб семафор не налага такъв строг ред на събуждане. При него, когато се извика операцията signal(), може да бъде избран произволен процес от блокираните, без да се спазва реда на пристигане. Това може да доведе до неравномерно разпределение на ресурсите и потенциално до ситуация на глад (starvation) за някои процеси.

Тема 7

- ✓ *Взаимно изключване - допускане само на един процес до общ ресурс.*
- *Опишете решение със семафори.*
- *Качества и свойства на конкретните файловите системи, реализирани върху block devices.*
- *Ефективна реализация, отлагане на записа, алгоритъм на асансьора.*

Взаимното изключване означава, че само един процес може да използва критичната секция от началото до края, докато останалите процеси трябва да изчакат. Това се постига чрез различни алгоритми – например алгоритъма на Петърсен – както и чрез хардуерни решения, внедрени в съвременните компютърни архитектури.

Общо взето, за еднопроцесорните системи се използват два метода:

Метод чрез забрана на прекъсванията:
При този подход в началото на критичната секция се забраняват всички прекъсвания, за да се осигури, че само текущият процес има достъп до ресурсите. В края на критичната секция прекъсванията се разрешават отново. Този метод беше типичен за първите еднопроцесорни операционни системи.

Spinlock (за многоядрени системи):
В съвременните паралелно работещи ядрови системи методът с забрана на прекъсванията не е приложим. Вместо това се използва допълнителен бит – наречен lock – който показва дали данните в критичната секция са свободни или се използват. При този подход процесът, който иска достъп до критичната секция, извиква процедурата spin_lock, която чрез операцията test-and-set проверява и "заключва" ресурса. След приключване на критичната секция ресурсът се освобождава чрез spin_unlock.

Пример за псевдокод:

```
spin_lock:
  disable_interrupts()     // Забраняване на прекъсванията
  R = test_and_set(lock)
  if R == 0 then goto critical_section
  enable_interrupts()      // Разрешаване на прекъсванията
  goto spin_lock
critical_section:
  // Изпълнение на критичната секция
spin_unlock:
  lock = 0
  enable_interrupts()      // Разрешаване на прекъсванията
```

Важно е този метод да не бъде рекурсивен и да се избягват прекъсвания по време на изпълнението на критичната секция, особено при драйвери, където се задават конкретни правила за управление на прекъсванията. Това е единствения механизъм от ниско ниво и е сравнително бърз. Недостатък е, че докато единия процес работи по критичната секция, другите процеси циклят, по-разумно е да бъдат спрени временно.

Тема 8

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

Тема 9

- *Взаимно блокиране (deadlock)*
- *Гладуване (livelock, resource starvation)*
- *Пример: задача за философите и макароните*
- *Единна йерархична файлова система в UNIX.*
- *Файлове и директории, команди - cd, mkdir, rmdir, cp, mv, rm*

Гладуване е просец, които чака дълго време.

Тема 10

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

- ✓ *Опишете функционалността на следните команди в Linux:*
  ls, who, find, ps, top

ls - Извежда списък на файловете и директориите в текущата или посочена директория.

> **ls** [-опции] [аргументи]
> Опции :
> - -l – в дълъг формат
> - -a – показва и скритите файлове
> - -r – в обратен ред
> - -t – по време на създаване
> - -d - за директории

who - Показва списък на потребителите, които в момента са влезли в системата.

> who –u – командата предоставя допълнителна информация за всеки потребител, включително времето на влизане, терминала, от който е влязъл, и периодът на неактивност

find - командата без аргументи извежда на екрана абсолютно всички файлове (дори и тези в поддиректориите) на текущата директория, в която се намираме

> Синтаксис: **find [dir-to-be-searched] [arguments]**
> Аргументи:
>
> - **-name** - търси по име на файл
> - **-type** - търси по тип **f** или **d**
> - **-user** - търси по притежател на файла
> - **-group** - търси по групата, притежател на файла
> - **-size** - търси по размер на файла (трябва да специфицирате мерната единица) - **c**
> - **-exec** - изпълнява команда върху всеки един намерен резултат

ps – Командата ps служи за извеждане на информация за активните процеси в системата. С флага -a се показват всички процеси, стартирани от текущия потребител (в някои реализации – всички процеси, които имат терминал). За да пренасочите изхода на ps към файл, можете да използвате оператор за пренасочване:

> $ ps -a > demo.txt
> Тази команда записва списъка с процеси във файла demo.txt.
> За да филтрирате изхода на ps и да покажете само процеси, свързани с "gnome", можете да използвате конвейер с командата grep:
> $ ps -a | grep gnome
> Конвейрът (pipeline) позволява няколко програми да работят паралелно, като изходът на една команда става вход на следващата. Това се реализира чрез създаване на междинен комуникационен канал (pipe) и форкване на процеси, които да изпълнят отделните команди

top - Предоставя динамичен, обновяващ се в реално време изглед на работещите процеси и използваните системни ресурси.

Тема 11

- *Възможни състояния на процес. Механизми и структури за приспиване/събуждане.*
- *Диаграма на състоянията и преходите между тях.*
- *Опишете функционалността на следните команди в Linux:*
- *vi, tar, gcc*

Тема 12

- *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).*
- *Опишете функционалността на следните команди в shell:*
- *echo, read, test, if, for, while*

Тема 13

- *Таблици за съответствието виртуална/реална памет.*
- *Ефективна обработка на адресацията - MMU, TLB.*
- *файлови дескриптори, номера на стандартните fd, пренасочване*
- *филтри - cat, grep, cut, sort, wc, tr*

Файловите дескриптори са локални за всеки процес. В UNIX системите те се представят като малки последователни числа, започващи от 0, и служат за управление на комуникационните канали, чрез които процесът чете или записва данни.

Обикновено се използват три стандартни файлови дескриптора, свързани с два основни канала:

- **Файлов дескриптор 0 (stdin):**
  Стандартният вход, който обикновено е свързан с клавиатурата (или с виртуално устройство, ако се симулира клавиатура).
- **Файлов дескриптори 1 и 2 (stdout и stderr):**
  Те представляват стандартния изход и стандартната грешка, съответно – обикновено свързани с терминала.

Това наследяване позволява процесът, който създава нов процес (чрез fork или подобна функция), да предаде предварително настроената среда за работа. Например, чрез пренасочване на входа и изхода може да се подготви среда за изпълнение на команда с определени настройки.

Пренасочването се осъществява с помощта на специални символи, които се използват в командния интерпретатор (shell):

- – Пренасочва стандартния изход към даден файл (замества съдържанието му).
- **<** – Пренасочва стандартния вход от даден файл.
- **>>** – Пренасочва стандартния изход към файл, като добавя към съществуващото съдържание (append).
- **;** – Позволява последователното изпълнение на няколко команди, като ги разделя.
- **&** – Когато се постави в края на команда, изпълнява командата във фонов режим, позволявайки на потребителя да продължи да въвежда нови команди, докато процесът работи на заден план.

Тема 14

- *Избройте видове събития, причиняващи повреда на данните във файловите системи.*
- *Опишете накратко стандарта RAID5. Какво е журнална файлова система?*
- *Свързване и допускане до UNIX система - login.*
- *Конзола - стандартен вход, стандартен изход, стандартна грешка.*
- *Команден интерпретатор - shell. Изпълнение на команди, параметри на Команди*
  В някои по-примитивни системи самия команден интерпретор е част от ядрото. Съвременния shell е потребителска програма.

Тема 15

- *Опишете разликата между синхронни и асинхронни входно-изходни операции.*
- *Дайте примери за програми, при които се налага използването на асинхронен вход-изход.*
- *Опишете с по едно-две изречения работата на следните системни извиквания в стандарта POSIX:*
  socket(), bind(), connect(), listen(), accept()

Тема 16

- *Опишете понятието "пространство на имената" (VFS).*
- *Структура, обекти и техните атрибути във VFS за ОС Linux.*
- *Основни функции, които обслужва пространството на имената.*
- *Една от класическите задачи за синхронизация се нарича "Задача за читателите и писателите" (readers-writers problem).*
  Опишете условието на задачата и решение, използващо семафори.

Тема 17

- *Опишете какви атрибути имат файловете в съвременна файлова система, реализирана върху блочно устройство (block device).*
- *Опишете накратко целта и реализацията на следните инструменти:*
- *(а) разместване във времето на дисковите операции, алгоритъм на асансьора.*
- *(б) поддържане на буфери (кеширане) на файловата система.*

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

Тема 18

- *Опишете накратко основните комуникационни канали в ОС Linux.*
- *Кои канали използват пространството на имената и кои не го правят?*
- *Опишете какви изисквания удовлетворява съвременна файлова система,*
- *реализирана върху блочно устройство (block device).*
- *Опишете предназначението на журнала на файловата система.*