1. Неформални описания за операционната система.
Софтуерът, който организира работата на дадена изчислителна система. Целта ѝ е да я поддържа и да ѝ даде възможно най-много функционалности, възможности за работа.

1.1. Посредник между агентите, ползващи изчислителната система
Файловете; драйверите; логическото устройство (операционната система) представляват посредници между агентите използващи изчислителната система

1.2. Управител на ресурси
Можем да кажем, че операционната система играе ролята на управител на ресурсите на изчислителната система и удобното им представяне за различните агенти (хора, програми и т.н.).

Операционната система трябва ефективно да управлява ресурсите на машината, като ги разпределя между много програми и потребители, състезаващи се за правото да ги използват. Например, ако няколко програми(процеса) се опитат едновременно да извеждат на едно и също печатащо устройство. Какво ще се случи? ОС трябва да осигури използването на този ресурс по такъв начин, че едновременно изпълняваните програми да не си пречат. Многопотребителската ОС поддържа едновременна работа на няколко потребителя. Тогава управлението на ресурсите става още по-важна функция, защото последователността, в която различните потребители ще използват ресурсите е непредсказуема. ОС трябва да решава от кого да се използва определен ресурс при наличие на няколко заявки за него, т.е. да разпределя ресурсите. Сложността на тази задача идва от там, че ресурсите са различни от гледна точка на възможностите за съвместното им използване. Кои са ресурсите? Това са процесор, оперативна памет, входно-изходни устройства.
1.3. Среда,осигуряваща стандарти и инструменти за комуникация между агентите на ИС

Една от първите услуги, които възникват като част от ранните операционни системи, е да се направят единни инструменти за работа с хардуера или поне за цели класове устройства - четене/писане върху магнитна лента, работа с диск. Този тип дейност наричаме абстрахиране на хардуера - т.е. заменяне на реалния хардуер с друго, виртуално устройство, което веднъж да е добре дефинирано и да ползваме само него, а самата работа с хардуера да я върши софтуер, който вече имаме и е част от операционната система.
2. Основни абстракции
2.1.Файл
обособен масив от байтове, който си има име и може да бъде достъпен от другите агенти в системата. Загубеният файл не съществува.;

Осигурява унифицирани операции за вход и изход, т.е. операции, които не зависят от входно-изходните устройства, съхраняващи данните.
Файловата система представлява съвкупност от файлове и информация, която описва определени характеристики на всеки от тях. Целта на файловата система е да позволи

подредба на файловете с цел по-лесно откриване. Има много файлови системи — те се различават по начина, по който се осъществява подредбата на файловете и по допълнителните характеристики, които се пазят за всеки файл. Тези характеристики могат да включват вид, права за достъп, размер, дата на последна промяна и др. Файловите системи обикновено подреждат файловете в структура, наречена Файлово дърво. Има различни файлове според функциите си във файловата система. Съвременните файлови системи поддържат съответствие кой е собственика, кой има права на достъп до файла(права и привилегии).

2.2.Процес

В операционните системи понятието процес е централно, всичко останало, включително и абстракцията файл, се гради върху него. Съвременните компютри са способни да изпълняват няколко операции едновременно. Докато централния процесор (ЦП) изпълнява команди, дисковото устройство може да чете и терминалът или принтерът да извеждат данни.

2.3.Комуникационни канали

За да може процесите да си комуникират помежду си, те използват операционната система като главен канал за тази връзка. Използват се за да се доберем до други ресурси извън локалното простанство. Самият канал е мястото, което е достъпно за двата процеса.

2.4. Имена

осъществяването на връзка(адресация) между обекти, ресурсите трябва да бъдат именувани.

Файлов дескриптор - Неотрицателно цяло число, което служи за уникален идентификатор на отворен файл. При отваряне на файл, ядрото извършва необходимите действия за отваряне на файла, зарежда информация за него. След това връща файловия дескриптор, който се свързва с отворения файл, ползва се в програмата, а след затваряне се освобождава. Дескрипторът има локално значение - връзката между дескриптор (число) и отворен файл важи само за текущия процес.

2.5.Хардуерните абстракции

Харудерните абстракции често позволяват на програмистите да пишат приложения, които са устройство-независими и са високо-производителни. Това става като се правят стандартни извиквания от операционната система към хардуера.

2.6.Приложно-програмният интерфейс

Приложно-програмният интерфейс е интерфейсът на изходния код, който операционната система или нейните библиотеки от ниско ниво предлагат за поддръжката на заявките от приложния софтуер или компютърните програми.

Образно казано, приложно-програмният интерфейс предоставя един по-абстрактен и опростен план за разработчика на приложения, който би му спестил изучаването на няколко различни слоя от Операционната или софтуерната система зад интерфейса. По този начин се достига ефективност и бързина при адаптирането на нови софтуерни

технологии. В миналото терминът се е използвал за обозначението на интерфейса между две програми.

3. Разпределено ползване на ресурсите
3.1. Разпределяне на ресурса на части

Ресурса е споделен или мултиплексиран между потребителите. Това може да е времево-мултиплексиране(потребителите се редуват) или място-мултиплексиране(всеки потребител получава част от ресурса)

3.2. Разпределяне във времето

Програмите или потребителите (всъщност процесите) използват ресурса последователно една след друга. Задачата на ОС е да решава: Кой да е следващия? Колко дълго да го използва? Да отнема ли насилствено предоставен ресурс? Когато ОС насилствено отнема ресурс от процес, казваме че има преразпределяне. Пример за разпределяни по този начин са:
Централен процесор – обикновено с преразпределяне;
Печатащо устройство – обикновено без преразпределяне.

3.3. Разпределяне в пространството

Ресура 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 ресура.
Ресурси,управлявани по този начин, са оперативна памет и дискова памет.

4. Видове ОС по начина на разделяне на изолация на ресурсите
4.1. Еднозадачна, еднопотребителска ОС

Еднозадачната ОС изпълнява само една задача в даден момент. Еднопотребителските ОС не различават потребителите, но не са непременно еднозадачни, тъй като няколко програми могат да се изпълняват в съчетание. Например MS DOS е еднопотребителска еднозадачна ОС.

4.2. Многозадачна ОС

Многозадачната изпълнява няколко задачи едновременно. Това се постига чрез разделение на времето на работа на процесора според инструкциите на специална подсистема (task scheduling subsystem). при разпределена многозадачност (pre-emptive multitasking) 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. Такива ОС са Unix-базираните като Solaris, Linux.
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 – такива са 16-битовите версии на Microsoft Windows.

4.3. Времеделене
Тъй като скоростта на въвеждането на данни от оператор е много по-ниска от скоростта на обработка на същите данни от процесора, използването на компютъра от само един оператор води до прахосване на скъпи изчислителни ресурси. „Разделението по време“ (time sharing) 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.

4.4. Многопотребителска ОС
Многопотребителските ОС разширяват 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.

5. Комуникация и общо ползване на ресурс
5.1. Съревнование на ресурси (race condition)
Съревнованието на ресурси е поведението, при което изхода е зависим от последователнноста или тайминга от други неконтролируеми събития.(когато два сигнала се състезават помежду си за да достигнат изхода първи);
Когато няколко процеса работят с един и същ ресурс по едно и също време.

5.1.1. Критична секция, атомарна обработка на ресурса
Тази част от програмата, в която се осъществява достъп до обща памет или се вършат неща, които могат да доведат до състезание се нарича критичен участък. Процесът е започнал и не е завършил изпълнението му, независимо от състоянието си.
Условия за избягване на състезанието
1.В един момент най-много един процес може да се намира в критичния си участък.
2.Никой процес да не остава в критичния си участък безкрайно дълго.
3.Никой процес, намиращ се вън от критичния си участък, да не пречи на друг процес да влезе в своя критичен участък.
4.Решението не бива да се основава на предположения за относителните скорости на процесите.

5.1.2. Защита на ресурса
Когато няколко процеса работят с един и същ ресурс по едно и също време. Решение -

обединяване на инструкциите на двата процеса в една единствена инструкция.

При еднопроцесорите машини това става сравнително лесно - временно се спират прекъсванията (докато се изпълни дадената инструкция докрай и да се постигне атомарност).

Друго решение (значително по-добро) е да се осигурят инструменти за защита на структурата данни. Например като към всяка такава структура добавим специален бит, който да показва дали е свободна, или не. Когато даден процес иска да работи с нея, първо проверява дали е свободна. Ако не е, изчаква, а ако е - запазва я и като приключи работата си с нея я освобождава.

5.1.3. Наредба във времето на процесите,ползващи ресурса
Непредсказуемо е времето,в което ще се изпълнят процесите.

5.2. Хардуерни инструменти за защита (lock) на ресурса
5.2.1. Enable/disable interrupt за еднопроцесорна система
Самите процедури за обработка на структурата да им осигурим изпълнението като единна инструкция неделима във времето. Инстукрцията 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. При еднопроцесорните системи се постига взаимно изключване просто като се изключат прекъсванията в критичния участък
5.2.2. test and set
Тази инструкция се използва, за да се пише в паметта и да се върне старата стойност като атомарна операция. ЦП може да използва test-and-set инструкция предложена от друг компонент,напривем двупортор RAM; ЦП може също да предложи test-and-set инструкция.
5.2.3. 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. Това е направено като единична атомарна операция.

5.3. Spinlock
Spinlock е ключ, който кара нишката,която се опитва да го вземе, да изчаква в цикъл,проверявайки постоянно дали ключа е свободен. Тъй като нишката остава активна, но не извърша смислена задача,това се нарича активно чакане.

6. Синхронизация от високо ниво - семафор
Когато два процеса си комуникират, те обикновено използват обща памет. При използването на обща памет се появява проблемът с надпреварата за ресурси (т.е. ако двата процеса използват паметта по едно и също време, те могат да развалят консистентността на информацията, съхранявана в нея). За да се реши този проблем, в съвременните операционни системи, зоните, в които тези процеси си комуникират, са изолирани от процеса и са предоставени на ядрото на операционната система. Например, работата по обслужването на тръбите (pipes) се извършва изцяло от ядрото.
Подобен проблем възниква и в многонишковото програмиране (когато към даден процес, предоставен от операционната система, паралелно се извършват няколко подпроцеса, които се налага да си комуникират помежду си). Друг случай, в който се използват общи ресурси и трябва да се синхронизира работата на програмите, е при базите от данни. При тях също се случва едновременно да работят няколко процеса, които да искат да достъпват и променят едно и също парче памет.

Когато опашката е опашка в обикновения смисъл (винаги елементи се взимат отпред, а новите се слагат отзад), тогава семафорът е силен (събуждането е в реда на заспиването). Слаб семафор е семафор, при който не е определен редът на събуждането. При него, ако има дълга опашка от заспали процеси, може да възникне "livelock" (процес, който е в операционната система, но не напредва своята работа - приспан е, чака някой да го събуди, но никой не го събужда) или "starvation" (процесът постоянно чака да му се дадат ресурси, но не му се дават и той гладува).

6.1 Приспиване и събуждане на процеси (block/ wakeup)
Операцията P проверява и намалява значението на брояча на семафора, ако това е възможно, в противен случай блокира процеса и го добавя в списъка на чакащи процеси, свързан със съответния семафор. Събитието, което блокирания процес чака, е увеличение на брояча на семафора. Това събитие ще настъпи когато друг процес изпълни операцията V над същия семафор. Операцията V събужда един блокиран по семафора процес, ако има такива, а в противен случай увеличава брояча на семафора, т.е. запомня едно събуждане. За операциите P и V се използва и обозначението down и up.

6.2 Семафор - дефиниция
Структура от данни, която е много полезна и приложима при решаването на синхронизационни въпроси.
Семафорът е като целочислена променлива S, приемаща само неотрицателни стойности, за която са достъпни две операции:
P(S) - Ако S>0, то S=S-1, иначе процесът, който изпълнява операцията се блокира.
V(S) - Ако има процеси, блокирани по семафора S, то един от тях се освобождава, иначе

S=S+1.

Когато нишка намалява стойността на семафора , ако семафора е отрицателен нишката блокира и не може да продължи докато друга нишка не увеличи семафора
Ако стойността на семафора е отрицателна и една нишка го увеличи , някоя от блокиралите нишки се събужда

Когато стойността на семафора е положителна, това значи колко нишки могат да го намалят преди да блокират , когато е отрицателна - колко нишки са блокирани , когато е 0 - няма нишки ,които чакат(няма заспали нишки) и ако нишка се опита да го намали ще го блокира

6.3 Реализация
6.4 Семафор като охранител на бариера (ресурс)
Бариера, синхронизираща два процеса:никой процес не може да мине след бариерата докато и двата не стигнат до нея. С всеки процес се свързва синхронизационен флаг, който се реализира със семафор.

7. Прости задачи за синхрозация чрез семафори:
7.1 Signaling (Инструкция от процес Q изчаква инструкция от процес P)

Thread P	Thread Q
p1	sem.wait()
sem.signal()	q1

7.2 Rendezvous - Взаимно изчакване (среща във времето)

Thread P	Thread Q
p1	q1
pArr.signal()	qArr.signal()
qArr.wait()	pArr.wait()
p2	q2

7.3 Mutex (взаимно изключване -допускане само на един процес до общ ресурс)

8. Още задачи за синхронизация чрез семаори:
8.1 Pipe

Програмният канал осигурява еднопосочно предаване на неформатиран поток от данни (поток от байтове) между процеси и синхронизация на работата им. Реализира се като тип файл, който се различава от обикновените файлове и има следните особености:
- За четене и писане в него се използват системните примитиви read и write, но дисциплината е FIFO.
- Каналът има доста ограничен капацитет.

8.2 Pipe с буфер - тръба съхраняваща n пакета информация. Използване на семафорите като броячи на свободни ресурси.
Пример: Програмата, която изпраща данни може да произвежда 5000 байта за секунда, а програмата която получава данни може да приема само 100 байта за секунда, но никаква информация не се губи. Информацията изпратена от изпращача се пази в буфер,който има определен размер. Когато програмата, която чете данни е готова за приемане ,започва да чете от буфера.

9. Проблеми при некоректна синхронизация:
9.1 DeadLock (Взаимно блокиране)
Казваме, че множество от два или повече процеса са в дедлок (deadlock), ако всички те са в състояние блокиран и всеки чака настъпването на събитие, което може да бъде предизвикано само от друг процес в множеството. Тъй като всички процеси са блокирани, никой от тях не може да работи и да предизвика събитие, което да събуди някой друг процес от множеството. Следователно, ако системата не се намеси, всички процеси ще останат вечно блокирани. Събитието, което процесите чакат най-често е предоставяне на ресурс на системата.

9.2 Livelock, resource starvation (Гладуване)
Този проблем се появява когато на процес не му бъде доставен достъп до нужните ресурси, без които този процес никога няма да може да завърши своята работа. Обикновено гладът е причинен от прилагането на твърде опростен алгоритъм за съставяне на разписания по които процесите се изпълняват. Този алгоритъм, който е част от ядрото (kernel), би трябвало да предоставя еднакво ресурси за всеки един от процесите, тъй че нито един от процесите в разписанието или разписанията да не остава без нужните ресурси.
Livelock е подобно на deadlock,обаче с разликата че състоянията на процесите,които участват в livelock-а постоянно се променят 1 с друг, като никой не прогресира.

9.3 Пример: задача за философите
Пет философа седят около кръгла маса, като пред всеки от тях има чиния със спагети, а между всеки две чинии има само по една вилица. Животът на всеки философ представлява цикъл, в който той размишлява, в резултат на което огладнява и се опитва да вземе двете вилици около своята чиния. Ако успее, известно време се храни, а след това връща вилиците.

Решение на задачата с философите - първо всички взимат дясната вилица. При опит да вземат и лявата, виждат, че е заета и оставят и дясната. След случайно време правят нов опит - взимат дясната и опитват да вземат и лявата. Ако е свободна, започват да ядат, ако не е - изчакват отново случайно време. Това решение предотвратява deadlock-а, но може да доведе до starvation.
Друго решение е общите ресурси да бъдат номерирани и когато трябва да вземем няколко ресурса, всеки процес да ги взима в нарастващ ред. Това решение също не гарантира, че няма да се получи starvation. Също така не работи в случаите, когато не знаем отначало точно кои ресурси ще ни трябват, а разбираме постепенно в процеса на работа. Тогава се прилагат други техники.

10. Процеси и тяхното управление:
Състоянията изразяват моментната необходимост на процеса спрямо неговата нужда от изчислителен ресурс.

10.1 Процеси в многозадачната система

Процесът е работеща програма, съществуваща във времето, 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).

10.2 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 вх/изх операция(wait-предизвиква се от синхронизиращия механизъм-семафора, семафорът приспива процеса, приспива само ако ресурсът е зает) или ако чака момент от времето(sleep-предизвикан от синхронизираща операция, приспива задължително)
- Спящият процес може да премине в активен(събужда се). Събуждането на процеса се предизвиква от завършването на входно-изходна операция на друг процес, който чрез семафор signal() ще подаде сигнал, че е освободен ресурс. 9

10.3 Превключване в система с времеделене - timer interrupt (часовник/scheduler)
Работещия процес може да стане блокиран (да му се отнеме процесорното време), ако твърде много време е стоял в процесора. В такъв случай времето му изтича, той бива прекъснат от timer, за да освободи CPU ресурс. Това е така, защото в съвременните ОС времето го делим на части и всеки процес има макс 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-това става, когато му се предостави процесорно време. Извършва се смяна на работещия процес, поради изтичане на време

на даден процес, ядрото тогава решава, кой чакащ процес да заработи(зависи от алгоритъма на ядрото и то неговия task scheduler).

- Когато спящият процес е приспан заради очакване на момент от времето(sleep-приспиването става по желание на процеса), то той може да стане активен, като този процес се инициира от timer(прекъсване на часовника).

11. Състояния на процеса и преходи между тях:
11.1 Възможни състояния и места за съхранение на съответните процеси
11.2 R - работещ (заема процесор)

Процес започва жизнения си цикъл в R “бягащо” състояние и завършва след като родителския процес вземе изработеното му от Z “зомби” състояние.
11.3 r - активен (чака за процесорно време) , run_queue
Активни - които чакат да се освободи процесорно време

В многозадачна ОС много процеси се изпълняват в едно и също време. Активните процеси се съхраняват в структура от данни наречена (масив) run_queue. Run queue съдържа приоритетните стойности за всеки процес,
които ще бъдат използвани от scheduler, за да определи кой ще бъде следващия процес ,който ще се изпълни. За да се подсигури , че всяка програма има правилно разпределение на ресурсите, всяка една се изпълнява за определено време ,след което се паузира и се слага в run_queue. Когато програмата е прекъсната, за да се изпълни друга, програмата с най-високия приоритет в опашката се изпълнява.
Процесите/програмите се махат от run_queue когато се приспят , когато чакат да се освободи или когато са приключили работата си.

11.3 S - спящ (чака събие, signal() от друг процес или драйвер) ,
входно-изходен канал

Процесът изчаква някакво събитие да се случи(като изпълнението на входно/изходна операция) и след това започва своето действие
11.4 T - чака за настъпването на момент във времето, time_queue
Процесът е зареден в главната памет и изчаква изпълнение върху ЦП. Модерните компютри имат възможността да подкарат множество различни програми или процеси по 1 и също време. Обаче ЦП може да обработва само по 1 процес. Процеси, които са готови за обработка са държани в опашка.

11.5 Диаграма на състоянията и преходите между тях

СФ-Системна Фаза,ПФ-Потребителска Фаза
12. Управление на паметта
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.

12.2 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.

12.3 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:

o 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):

- 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;
- Има парчета за споделени библиотеки, които са достъпни едновременно за няколко процеса и се предоставят на процеса;

o Динамична част(read-write памет)

- Стек - 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.
- Heap - памет, която се използва от процеса, ако е необходимо да се задели допълнително обеми памет, които не са в локалните променливи(използва се при нужда и тя варира от процес до процес, блок с неопределена дължина)
- Блок за статична памет - променливи, които при стартиране на процеса се заделя винаги тази памет; място за данни-обем на масиви, константи и др.

Структурата на виртуалната памет може да се разгледа по следния начин:
Данни - Глобални променливи, константи, статични променливи от програмата
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.
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).
Heap - Използва се за някои видове работни хранилища.
Споделени библиотеки - Споделените библиотеки са позиционно независими и така те могат да бъдат споделени от всички програми, които искат да ги използват.

13. Виртуална памет- реализация

13.1 Таблици за съответствието на 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.

13.2 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 във физически адрес.

Модерните MMU по принцип 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.

Translation lookaside buffer) е 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 преобразува виртуалните адреси в сегментни адреси.

14. Синхронен и асинхронен вход/изход

14.1 Опишете разликата между синхронни и асинхронни входно-изходни операции

Разликите в синхронните и асинхронните входно-изходни операции са базирани върху начина по който те организират входно-изходните процеси.

При Асинхронните входно-изходни операции или още interrupt driven, целта е да да има възможно най-малко приспани процеси. Този вид операции позволяват на процесите да продължат докато предаването на данни и изпълнението на други процеси приключи и се освободи място. При постъпване на процес се изпраща запитване до ядрото за изпълнение на процеса и ако то се отхвърли и няма възможност за изпълнение на процеса, то той се отпраща, а ако се приеме той се пропуска.

За разлика от Асинхронните входно-изходни операции, Синхронните организират процесите по по-различен начин. При Синхронните входно-изходни операции, процесорът се обръща към контролера на входно/изходното устройство и подава заявка за извършване на операцията. Тук има повече приспивания и събуждания на процеси. При постъпване на процес, ако не е възможно да се изпълни, той се приспива, докато не се освободи място за неговото изпълнение.

14.2 Дайте примери за програми, при които се налага използването на асинхронен вход - изход

Пример за асинхронен вход/изход е pipeline - това е като тръба, по която могат да минава информация само в едната посока.

Втори пример:

SQLite използва асинхронен вход/изход. Попринцип,когато SQLite пише в файл от базата данни, изчаква докато операцията за писане е приключила преди да даде контрол на извикващото приложение. Тъй като писането във файловата система обикновено е много бавно в сравнение с ЦП операции, това може да създаде bottleneck при производителността. Асинхронният вход/изход бекенд е разширение, което кара SQLite да извърши всичките заявки за писане използвайки отделна нишка, която работи на заден план. Независимо, че това не намалява използването на системните ресурси(ЦП,скорост на диска), то позволява на SQLite да върне достъп на извиквача бързо дори при писане в базата от данни.

---- край на част 1 -----

ТЕМИ, специфични за Linux/UNIX

15. Текстова конзола, shell

//В shell-а може да пишем команди, които се интерпретират от операционната система

15.1 Свързане и допускане до UNIX система- login

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

15.2 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.

15.3 Shell - команден интерпретатор

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)

15.4 Изпълнение на команди, параметри на команди

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

16. Shell - конвейри , пренасочване , филтри
16.1 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.

16.2 Филтри - cat, grep, cut, sort, wc, tr
Cat - обединява съдържанието на файловете и го извежда в терминала
Grep - филтър на ниво редове (-c-на колко реда е match-нато условието, -v-обръща условието, -i-case insensitive match, -n-показване номера на реда, -in-съдържа, --color-оцветява match-натото, -A/-B/-C n - показват се n реда след/преди/преди и след match-а)
Cut - филтриране на ниво колони(-d-разделител, -f-номер на желания field (1,2 -> първа и втора колона, 1-5 -> от първа до пета,1- ->от първа до края), -c-character)

Sort - сортира редове на текстови файлове(-t-указване на разделител, -k-номер на желаната колона, -n-сортиране като числа, а не лексикографски, -r-сортиране в обратен ред)

Wc - статистика за текстови файлове(-c-брой байтове, -m-брой character-и, -l-брой редове, -w-брой думи)

Tr - translate,squeeze and delete character(-d-изтриване, -s-squeeze, където има повече от едно срещане, ги свива до 1)

17. Shell - програми (скриптове)
17.1 echo (-n - no newline, -e - указва на команда echo да разбира от специални символи като \t и \n)
17.2 read - четене на потребителски вход
One line is read from the standard input, and the first word is assigned to the first name, the second word to the second name, and so on, with leftover words and their intervening separators assigned to the last name.

If there are fewer words read from the standard input than names, the remaining names are assigned empty values.

17.3 test - всички условия в if ()
-числово сравняване(-eq-equal, -ne-not equal, -gt-greater than, -ge-greater than or equal, -lt-less than, -le-less than or equal)

сравняване на низове(=, !=, >, >=, <, <=)
-логически сравнения(-o-or, a-and, &&, ||)
-f-дали даден файл съществува и е обикновен, -d-дали съществува и е директория, -e-дали файл съществува, без значение какъв е, -z,-n - дали даден файл/променлива е с празно/непразно съдържание, -r,-w,-x - дали файл има права за четене/писане/изпълнение

17.4 if [...] then ... elif [...] then ... else ... fi

17.5 for - for ... in ... do ... done
17.6 while - while [...] do ... done

18. VI, tar, g++
Vi- Command mode − This mode enables you to perform administrative tasks such as saving files, executing commands, moving the cursor, cutting (yanking) and pasting lines or words, and finding and replacing. In this mode, whatever you type is interpreted as a command.
Insert mode − This mode enables you to insert text into the file. Everything that's typed in this mode is interpreted as input and finally it is put in the file .

tar - tape archive; команда за правене на архиви
Най-ранната функционалност на tar е просто да обедини няколко файла в един, по-голям файл, без да ги компресира по какъвто и да било начин. Това става като направи каталог, в който пише кой файл как се казва и колко е голям, а след него следват самите файлове.

tar [опции] [списък от имена на файлове]
tar cf archive.tar file1 file2 - създава архив с име archive.tar, който съдържа file1 и file2

Има два вида компресиране на данни - със и без загуба на информация.
z - като част от опциите указва на tar да компресира данните, не просто да ги обедини в по-голям файл
c - create, създаване на архив
t - показва съдържанието на архива
x - extract, разархивира

g++ - g++ is a *nix-based C++ compiler usually operated via the command line.
The default executable output of g++ is "a.out". It is also possible to specify a name for the executable file at the command line by using the syntax
-o outputfile

19. Програми за разглеждане и търсене
19.1 ls (list directory contents) имената на файловете и директориите в текущата директория (-l-допълнителна информация, -a-показва и скритите файлове, започващи с ".", -dl-показва само директорията без съдържанието й)
19.2 who - текущо влезлите потребители в системата
19.3 find - търси файлове в йерархията на директориите
19.4 ps -(process state) - текущите активни процеси(-ef-списък от всички процеси)
ps -aux - абсолютно всички процеси
19.5 top - показва изглед на стартираните процеси в реално време. Може да сортира процесите по CPU usage, memory usage и runtime
-p - показва само процесите с дадено process id

20. Файлова система
20.1 Единна йерархична файлова система в UNIX
20.2 Файлове и директории, команди - cd,mkdir, cp ,mv, rm
Ch - change directory
Mkdir - make directory(-m-set permissions, -p-parent directories)
Cp - създава копие на файл/файлове

Mv - преименува и мести файлове/директории
Rm - трие файлове/директории(-r-премахва директории и съдържанието им рекурсивно)

21. Многопотребителска работа
Многопотребителските ОС разширяват 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.

21.1 Права и роли в UNIX
Всеки потребител има user ID (uid). За всеки файл в системата е дефинирано кой е собственикът (кой user). Има и групи от user-и. За всеки файл е казано: собственикът, членовете на неговата група и останалите хора какво могат да правят с него.
ls -l - първите символи указват правата за достъп до него; 1 - вид на файла; 2, 3 и 4 - права на собственика; 5, 6 и 7 - права на групата; 8, 9 и 10 - права на всички останали; Тройките представляват 3 отделни бита. Първият може да е - или r, вторият - да е - или w, третият - да е - или x. Отговарят съответно за четене, писане и изпълняване на файл. Правата за достъп могат да се променят от собственика на файла или от супер потребителят. Това става чрез chmod.

21.2 Права - u/g/o user/group/other
21.3 Роли r/w/x read/write/execute
21.4 chmod - (change file mode bits) - добавяне/премахване на права

22. Физически файлови системи
файлът е обект, който съхранява информация, не прилича на процеса. Ние ще приемем само, че файлът е абстракция, която ни предоставя операционната система и която ни дава възможност дълготрайно да съхраняваме информация и също комуникира с останалите обекти.

Файловата система представлява съвкупност от файлове и информация, която описва определени характеристики на всеки от тях. Целта на файловата система е да позволи подредба на файловете с цел по-лесно откриване. Има много файлови системи — те се различават по начина, по който се осъществява подредбата на файловете и по допълнителните характеристики, които се пазят за всеки файл. Тези характеристики могат да включват вид, права за достъп, размер, дата на последна промяна и др. Файловите системи обикновено подреждат файловете в структура, наречена Файлово дърво. Има различни файлове според функциите си във файловата система. Съвременните файлови системи поддържат съотвествие кой е собственика, кой има права на достъп до файла(права и привилегии).

22.1 Блочни и символни устройства
Блочни специални файлове или блочни устройства предлагат буфериран достъп до хардуерни устройства и предлагат някаква абстрацкия от тяхната специфика. За разлика от символните устройства,блочните устройства винаги ще позволят на програмиста да чете или пише блок от всякакъв размер и подравняване. Отрицателното на това е,че тъй като блочните устройства са буферирани, програмистът не знае колко време ще отнеме записаните данни да се прехвърлят от кернела в истинското устройство или в какъв ред 2 отделни записа ще пристигнат във физическото устройство.
??
Символно устройство е такова устойство с което диска комуникира чрез изпращане и получаване на информация байт по байт. Пример са звуковите карти, USB портовете , parallel ports
Блочното устройство комуника като изпраща цял блок от данни. Пример са твърдите дискове, USB камерите,

22.2 Конкретни файлови системи и точки за монтиране
22.3 /etc/fstab, /etc/mtab, mount, df, umount
/etc/fstab - системен конфигурационен файл. Този файл изкарва списък с всички налични дискови партишони и други типове на файлове системи,които не трябва задължително да са дисково-базирани, и показва как трябва да се инициализират или интегрират в по-голямата файлова-системна структура.
/etc/mtab - системен конфигурационен файл. Този файл изкарва списък с всички текущо монтирани файлови системи заедно с техните инициализиращи опции. Mtab прилича на fstab, разликата между двете е,че /dev/fstab изкарва списък с кои налични файлови системи трябва да се монтират по време на буутването,докато първото показва кои са монтирани в момента.
Mount - инструктира операционната система, че файловата система е готова за използване и я асоциацира с отделна точка в общата йерархия на файловата система(mount point) и задава опции свързани с нейния достъп.
Umount - инструктира операционната система, че файловата система трябва да бъде деасоциирана от точката и за маунтване, правейки я вече недостъпна и може да бъде изтрита от компютъра. Важно е първо да се umount-не устройство преди да се изтрие.

23. Физически файлови системи - реализация
23.1 Качества и изисквания към файловите системи
23.2 Ефективна реализация, отлагане на записа , алгоритъм на асансьора

Има специални програми (check and repair), които се стартират когато е ясно, че има някакви нарушения в структурата на файловата система, и които проверяват съответствието между метаданните и данните на целия диск. Когато системата има много процеси и едновременно се работи с много файлове, може да се окаже, че тези програми не могат да оправят нещата.

Всички съвременни файлови системи са журнални (Journal File System). Концепцията за журналност е въведена първо при базите данни - за транзакциите, които са одобрени и които трябва да се запишат в дисковите пространства, за да бъдат съхранени дълготрайно. За да не се загубят при авария, освен нормалния файл, в който са записани таблиците на базите данни, се съхранява и друг файл, който се нарича журнал (log file). Той е един за цялата база и всяка транзакция се записва първо в журнала. Извърши ли се дадена транзакция изцяло, тя се изтрива от журнала. Тази концепция се използва и при файловите системи - някакъв специален файл (или част от диска) се обявява за журнал и отложените операции се записват в правилния ред в журнала. Докато присъстват там се опитваме да ги запишем в реалните места на диска. Ако успеем, цялата поредица от действия, които са свързани с реалната промяна по даден файл, се изтриват от журнала. В различните журнали се описват различни типове данни.

Алгоритъм на асансьора
Алгоритъмът, който се използва най-често, се нарича алгоритъм на асансьора и предполага, че близки сектори на диска ще се променят бързо. При него имаме указател, който сочи къде се намира главата на твърдия диск в момента, помни се и посоката, в която се движи тя. Заявките се обработват по посока на движение на главата подобно на движението на асансьор - слиза надолу докато изпълни всички заявки, след това обръща посоката и се движи нагоре докато изпълни всичко. Недостатъкът на този алгоритъм е, че може да доведе до голямо забавяне на заявките. Друг алгоритъм е алгоритъмът на най-близкия сектор, който оправя недостатъка на алгоритъма на асансьора, но при него може да се получи starvation.

24. Специални файлове
24.1 Външни устройва и тяхното именуване (/dev, mknod)
The /dev directory contains the special device files for all the devices. The device files are created during installation
/dev -съдържа драйверите на системата; не съдържа файлове, в нея се описват устройствата, които са част от изчислителната система; псевдо файлове, които задават какъв хардуер може да бъде обслужван от нашата система и какъв точно се обслужва
mknod - системно извикания mknod() създава файлово-системен възел(файл,специален файл или именуван pipe) именуван pathname, с атрибути специфицирани от mode и dev.
The system call mknod() creates a filesystem node (file, device
special file, or named pipe) named pathname, with attributes
specified by mode and dev.
Mknod попринцип се е използвал, за да се създадат символните и блочните устройства,които се намират в /dev/. Напоследък софтуер като udev автоматично създава и премахва устройствени възли на виртуалната файлова система, когато съответстващият хардуер е открит от kernel-а.
mknod was originally used to create the character and block devices that populate /dev/.

Nowadays software like udev automatically creates and removes device nodes on the virtual filesystem when the corresponding hardware is detected by the kernel, but originally /dev was just a directory in / that was populated during install.

24.2 Линкове - твърди и символни, команда ln
Символният линк е прякора на всеки файл, който съдържа референция към друг файл или директория в формата на абсолютен или относителен път.
Твърдият линк е запис на директорията, който асоциацира име с файл във файловата система. Всички директории-базирани файлове системи трябва да имат поне 1 твърд линк давайки оригиналното име за всеки файл.
ln [OPTION]... TARGET [LINK_NAME] - създава линк към специфицирания TARGET с допълнителната настройка LINK_NAME. Ако LINK_NAME е пропуснат, линкът се създава със същото базово име като TARGET-а в текущата директория.

Symbolic link contains information about the destination of the target file.
The important part is that hard link is closely tied together with its originating file. If you make changes to a hard link, you automatically make changes to the underlying file that the hardlink is attached to.
Hard link can only refer to data that exists on the same file system.

Ln - s = create symbolic link, otherwise is hardlink
24.3 сокети
socket - двупосочен комуникационен канал; socket listen и socket connect; тези връзки могат да се създават своевременно и процесите от двете страни не е необходимо да са наследници на общ родител

25. Файлова система
25.1 Стандарно разполагане на файловете с Linux :
/bin -съдържа изпълними файлове;
важни програми, които се изпълняват при стартирането на системата, общодостъпни програми
/boot - информация за ОС преди самото й стартиране; неща, свързани със зареждането на операционната система
/dev -съдържа драйверите на системата;
не съдържа файлове, в нея се описват устройствата, които са част от изчислителната система; псевдо файлове, които задават какъв хардуер може да бъде обслужван от нашата система и какъв точно се обслужва
/etc -съдържа конфигурационните файлове; файлове, които управляват конфигурацията на конкретната изчислителна система след зареждането на операционната система; информация за потребителите, пароли и т.н.
/home -home директориите на потребителите в системата; файловете на конкретните потребители
/lib - library files, включително необходими драйвери за boot на системата

; важните библиотеки
/media - директория за монтиране на хардуерни у-ва(CD,DVD,flash); точка за монтиране на хардуерни инструменти, които невинаги са налични (USB памети, CD, DVD, ...)
/proc - информация за операционната система в реално време; подобна на /dev, не съдържа файлове, съдържа псевдо файлове със статистическа информация от ядрото. Начин за програмите да видят какво стават в ядрото (top командата например ги използва)
/root -home директорията на администраторския акаунт; home директорията на super user-a
/sbin - административни binary файлове(mount,shutdown,umount..); подобна на /bin, системен bin, не може да се използва от "нормалния" потребител, само от super user-a
/usr -програмите, достъпни при пълноценна работа на ОС;
 програми и неща, свързани с тях, които се използват по време на пълноценната работа със системата; всички програми, информация, общи за конфигурацията
/var -системни файлове - logging files, mail directories, printer spool..;
 за данни, не на потребителя, а за споделени данни (бази данни например), също невинаги е достъпна до обикновения потребител
/log - log файлове

26. API, POSIX, -работа с файлове
26.1 open()
close()
 read()
write()
 lseek ()- lseek(fd1,int offset,SEEK_SET); SEEK_SET премества указателя fd1 с offset;
SEEK_END
        The offset is set to the size of the file plus offset bytes.

 SEEK_CUR
        The offset is set to its current location plus offset bytes.

 scanf() - int scanf(const char *format, ...);
 The scanf() function reads input from the standard input stream stdin,
 fgets() -char *fgets(char *s, int size, FILE *stream);
ako imame da chetem string ove;
reads in at most one less than size characters from stream and
      stores them into the buffer pointed to by s. Reading stops after an
      EOF or a newline. If a newline is read, it is stored into the buffer.
      A terminating null byte ('\0') is stored after the last character in
      the buffer.

27 API,POSIX -работа с процеси и тръби
27.1 pipe(int pipefd[2])
creates a pipe, a unidirectional data channel that can be used
for interprocess communication. The array pipefd is used to return two
file descriptors referring to the ends of the pipe. pipefd[0] refers
to the read end of the pipe. pipefd[1] refers to the write end of the
pipe. Data written to the write end of the pipe is buffered by the
kernel until it is read from the read end of the pipe.

dup2(oldfd, newfd)- makes newfd be the copy of oldfd, closing newfd first if necessary

dup() - poluchavame kopie na pyrviq svoboden failove deskriptor

fork()

Създава нов процес(дете)

Ново- създаденият процес е почти идентичен на бащиния (този, който го е създал).

Защо почти? ами различават се например по PID (process ID, всеки процес има уникален PID)

Тъй като са почти идентични процесите, трябва някак да ги различаваме.

Различаваме ги по това какво връща fork().

fork() връща 0 при детето и > 0 при бащата.

exec()

int execl(const char *path, const char *arg, ..., 0);

първият аргумент е пълно име на бинарен файл (команда), който искаме да изпълним

вторият аргумент е името на самата команда

Следват N на брой аргументи - опции към командата, която искаме да изпълним

Последният аргумент е 0, указва край на въвеждане на аргументи.

Пример:

искаме да изпълним команда ls, с опция -l

execl("/bin/ls", "ls", "-l", 0);

Особености (които трябва да знаем):

При изпълнение на системния примитив execl се сменя образа на процеса, който я е изпълнил.

т.е. изпълни ли се execl -> сменя се текущата програма с програмата ls и връщане назад, няма.

Още примери - най-долу директно в задача.

Особености (които не е нужно да знаем):

Първият аргумент на функцията е бинарния файл, който искаме да изпълним.

Вторият аргумент е какво ще бъде името на стартирания процес, което може да видим с команда ps.

wait()

wait(status);

Тъй като процесите се изпълняват на конкурентен принцип, бащата може да завърши изпълнението си преди детето.

Но ако искаме изхода от детето да го ползваме в бащата, то трябва да укажем на бащата да изчака детето, да завърши.

Това става чрез системен примитив wait().

Той следи за промяна на статуса на детето:

Например настъпване на терминиране на процеса или на пауза или на събуждане от пауза.

waitpid()
The waitpid() system call suspends execution of the calling process
until a child specified by pid argument has changed state. By default,
waitpid() waits only for terminated children,

28.API,POSIX -сокети
socket()
bin()
connect()
listen ()
accept()