Основи на операционните системи

Какво е операционна система?

1. Software Layer (Софтуерен слой)

Операционната система (ОС) е основен софтуерен компонент на компютърната система, който действа като **слой между хардуера и приложенията**, които потребителите използват. Този слой **включва софтуерни инструменти и процеси**, които позволяват на компютъра да функционира ефективно и сигурно. Основната му цел е да осигури **по-лесен и сигурен достъп до хардуерните ресурси** (процесор, памет, дискове, устройства за въвеждане и извеждане) за потребителите и приложенията.

Основните функции на този софтуерен слой включват:

Управление на процесите: Процесите са основните работни единици в операционната система. Един процес представлява изпълнението на дадена програма и включва както кода на програмата, така и всички необходими ресурси, като памет и файлове.

Операционната система трябва да управлява множество процеси едновременно, осигурявайки, че всеки процес има достъп до необходимите ресурси и че се изпълнява без да засяга останалите. Това управление включва няколко ключови аспекта:

⇒ **Създаване на процеси (Process Creation)**: Когато стартирате програма, операционната система създава нов процес, като му предоставя уникален идентификатор (Process ID или PID). ОС резервира ресурси за процеса (памет, файлове, входни/изходни устройства и т.н.) и го добавя в списъка на активните процеси.

⇒ **Планиране на процеси (Process Scheduling)**: Планирането на процеси е механизъм, чрез който операционната система разпределя процесорното време между различните процеси. Има различни видове планиращи алгоритми:

- **Планиране със закъснение (Round Robin)**: Всеки процес получава фиксирано количество време за изпълнение (т.нар. квант време), след което е спрян и следващият процес получава своето време.
- **Приоритетно планиране**: Всеки процес има приоритет, и процесорът се предоставя първо на тези с по-висок приоритет.
- **Планиране с най-кратко оставащо време (Shortest Job First)**: Процесите с най-кратка продължителност се изпълняват първи.

⇒ **Прекратяване на процеси (Process Termination)**: Процесите могат да бъдат прекратени доброволно (когато приключат изпълнението си) или принудително (ако възникне грешка или е необходимо прекратяване от администратор или друга програма).

⇒ **Комуникация между процесите (Inter-Process Communication - IPC)**: ОС осигурява методи, чрез които различните процеси могат да комуникират помежду си, като

например използване на сигнали, съобщения или споделена памет. Тази комуникация е от критично значение за многофункционалните и мултитредингови приложения.

Управление на паметта: Паметта е един от най-важните ресурси на компютъра. Операционната система трябва да гарантира, че всяка програма получава достатъчно памет за работа и че няма конфликти между различните програми относно достъпа до паметта.

⇒ **Разпределение на паметта (Memory Allocation)**: ОС разпределя паметта за всеки процес, осигурявайки, че всеки процес получава необходимото количество памет за своите данни и инструкции. Има различни методи за разпределение:

- **Статично разпределение (Static Allocation)**: Разпределението на паметта се извършва по време на стартирането на програмата и не се променя по време на изпълнението.
- **Динамично разпределение (Dynamic Allocation)**: Паметта може да бъде разширена или намалена в процеса на работа на програмата, в зависимост от нуждите ѝ.

⇒ **Виртуална памет (Virtual Memory)**: ОС използва техника, наречена виртуална памет, за да осигури, че на програмите може да бъде предоставена повече памет, отколкото физически е налична в компютъра. Виртуалната памет позволява прехвърляне на данни между RAM и по-бавни устройства за съхранение (като твърдите дискове), като се създава илюзията за по-голяма оперативна памет.

⇒ **Странично подменяне (Paging)**: Това е техника, при която виртуалната памет е разделена на блокове (страници), които могат да бъдат премествани между физическата памет и вторичното съхранение. Това осигурява по-добра ефективност и използваемост на наличната памет.

⇒ **Защита на паметта (Memory Protection)**: Операционната система осигурява механизми, които гарантират, че един процес не може да чете или пише в паметта на друг процес. Това се постига чрез използване на хардуерни и софтуерни техники за защита.

Файлова система: Файловата система на операционната система е отговорна за организирането и управлението на данните, които се съхраняват на различни устройства за съхранение като твърди дискове, SSD и други. Файловата система осигурява интерфейс, чрез който потребителите и програмите могат да създават, четат, модифицират и изтриват файлове.

⇒ **Файлова структура**: Файловете се организират в йерархична структура, обикновено под формата на директории (или папки). Това позволява лесно намиране и управление на файловете.

⇒ **Типове файлови системи**: Има различни файлови системи в зависимост от операционната система и устройството:

- **FAT32**: Широко разпространена файлова система, често използвана на преносими устройства като флаш памети.
- **NTFS (New Technology File System)**: Файлова система, използвана в Windows, осигуряваща по-добра сигурност, компресия и възможности за възстановяване на файлове.

- **EXT4**: Файлова система, използвана в Linux, с добра производителност и поддръжка на големи файлове.

⇒ **Операции с файлове**: ОС предоставя функции за създаване, четене, писане, модифициране и изтриване на файлове. Тези операции могат да се извършват чрез системни повиквания (system calls), предоставени от ОС.

⇒ **Управление на права за достъп**: Файловата система също така осигурява механизми за задаване на права за достъп, като четене, писане и изпълнение на файловете. Това осигурява сигурност и защита на данните, като само определени потребители или програми имат достъп до дадени файлове.

Защита и сигурност: Една от най-важните задачи на операционната система е да осигури защита на данните и процесите от неоторизиран достъп и злонамерени действия. Защитата в ОС е съвкупност от механизми и политики, които предотвратяват неправомерно използване на системата.

⇒ **Удостоверяване (Authentication)**: Това е процесът на потвърждение на идентичността на потребителя. Обикновено се осъществява чрез въвеждане на потребителско име и парола, но могат да бъдат използвани и биометрични методи, смарт карти или двуфакторно удостоверяване (2FA).

⇒ **Авторизация (Authorization)**: След като потребителят е удостоверен, операционната система определя какви права за достъп има той върху системните ресурси. Това се извършва чрез политики за достъп, които задават кой може да чете, записва или изпълнява файлове и програми.

⇒ **Контрол на достъпа (Access Control)**: ОС прилага различни модели за контрол на достъпа, като:

- **Мандатен контрол на достъпа (Mandatory Access Control - MAC)**: Достъпът се базира на политиките, зададени от системните администратори.
- **Дискреционен контрол на достъпа (Discretionary Access Control - DAC)**: Собствениците на файлове или ресурси имат правото да определят кой може да има достъп до тях.

⇒ **Шифроване (Encryption)**: Шифроването осигурява защита на данните чрез преобразуването им в неразбираем формат, който може да бъде разшифрован само от упълномощени потребители или програми.

⇒ **Откриване и предотвратяване на атаки**: ОС може да открива и предотвратява различни типове атаки, като например атаки тип „отказ от услуга“ (DoS), вируси, троянски коне и други злонамерени софтуери.

Тези аспекти на защитата и сигурността са жизненоважни за защитата на чувствителни данни и гарантиране на стабилността на системата срещу потенциални заплахи.

2. Manages Computer Hardware (Управлява компютърния хардуер)

Операционната система играе ключова роля в управлението на хардуера. Тя осигурява интерфейс, който абстрахира сложността на хардуера и го прави достъпен за софтуера и потребителите.

- **Процесор (CPU)**: ОС разпределя процесорното време между различните задачи и процеси, чрез т.нар. "планиране на задачи". Това осигурява многозадачност и позволява множество програми да се изпълняват едновременно, без да пречат една на друга.
- **Памет (RAM)**: ОС управлява оперативната памет и осигурява, че всяка програма получава необходимото количество памет за изпълнение. Ако една програма използва твърде много памет, ОС може да наложи ограничения или да освободи ненужната памет.
- **Устройства за съхранение (HDD, SSD)**: ОС предоставя механизъм за четене, запис и управление на файлове и данни върху дисковите устройства.
- **Устройства за вход/изход**: Операционната система управлява взаимодействията с периферни устройства като клавиатури, мишки, принтери, дисплеи и други устройства.

3. Provides Services for Computer Programs (Предоставя услуги за компютърни програми)

Операционната система също така предоставя редица услуги, които програмите и приложенията могат да използват за изпълнение на задачи, свързани с взаимодействието с хардуера или други системни ресурси. Тези услуги включват:

1) **API (Application Programming Interface) – Интерфейс за програмиране на приложения:** представлява набор от правила и инструменти, които операционната система предоставя за разработка на софтуер. Чрез API програмистите получават достъп до системните ресурси, като памет, процесор, файлове, мрежи и периферни устройства. API осигурява стандартен начин за комуникация между софтуера и операционната система, без програмистите да се занимават с детайлите на хардуера.

    - **Системни повиквания (System Calls)**: Основният начин, по който програмите взаимодействат с операционната система, е чрез системни повиквания. Това са функции, предоставени от операционната система, които позволяват достъп до ресурси като отваряне на файлове, писане в тях, стартиране на нови процеси и т.н.
    - **Библиотеки**: ОС предоставя различни библиотеки, които опростяват използването на системните ресурси. Например, библиотеките за мрежови комуникации предлагат интерфейси за изпращане и получаване на данни по мрежи.
    - **Универсалност**: API осигурява стандартизиран интерфейс, което позволява различни програми да използват един и същ механизъм за достъп до ресурси, независимо от хардуера или специфичната конфигурация на системата.

2) **Управление на файлове:** Файловата система е отговорна за съхранението и управлението на данни във файлове и папки. Тя представлява логическа структура, която абстрахира физическото разположение на данните върху дискове или други устройства за съхранение.

- **Йерархична структура**: Файловата система организира файловете в йерархична структура с директории и поддиректории (папки). Това прави лесно намирането и управлението на файловете. Всяка директория може да съдържа множество файлове и други директории.
- **Операции с файлове**: ОС предоставя основни операции с файлове, като създаване на нови файлове, четене на съществуващи файлове, модифициране на файловете, както и изтриване. Всички тези операции се извършват чрез системни повиквания или чрез графичен интерфейс.
- **Разновидности на файлови системи**: Различните файлови системи осигуряват различни нива на ефективност, сигурност и съвместимост. Например:
    - **FAT32**: Поддържа устройства с малък капацитет, но има ограничение за размера на отделните файлове (до 4 GB).
    - **NTFS**: Осигурява по-добра сигурност и ефективност, и е предпочитаната файлова система за Windows системи.
    - **EXT4**: Популярна файлова система за Linux, която поддържа големи файлове и обеми на данни.
- **Журнализиране (Journaling)**: Много съвременни файлови системи (като NTFS и EXT4) поддържат журнализиране, което означава, че системата записва всички промени във файловете в специален дневник, преди да бъдат извършени. Това помага за възстановяване на данните в случай на системен срив или прекъсване на захранването.

3) **Управление на мрежи:** Мрежовите услуги на операционната система осигуряват комуникация между различни устройства и мрежови ресурси. Това включва както вътрешната комуникация между устройствата в локална мрежа (LAN), така и достъп до глобалната интернет мрежа.

⇒ **TCP/IP протокол: TCP/IP (Transmission Control Protocol/Internet Protocol)** е основният набор от комуникационни протоколи, използвани за предаване на данни по интернет и в локални мрежи (LAN). TCP/IP е сърцето на мрежовите комуникации и се състои от няколко нива, всяко от които изпълнява специфични задачи. Тези нива са:

- **Мрежов слой (Internet Layer)**: Това е слоят, който осигурява маршрутизация на данните между различни мрежи. Основният протокол на този слой е **IP (Internet Protocol)**, който гарантира, че пакетите данни достигат до своето предназначение. IP адресите играят основна роля тук, като те уникално идентифицират устройствата в мрежата.
- **Транспортен слой (Transport Layer)**: В този слой **TCP (Transmission Control Protocol)** и **UDP (User Datagram Protocol)** са основните протоколи.
    - **TCP** е по-надежден протокол, който осигурява сигурно предаване на данни, като се грижи за последователността на пакетите, проверката за грешки и повторното изпращане на изгубени пакети.
    - **UDP** е по-бърз, но ненадежден протокол, който изпраща данни без допълнителна проверка за грешки или гаранция за доставяне в

правилния ред. UDP се използва там, където бързината е по-важна от надеждността, като например при стрийминг на видео.

- **Мрежов достъп (Network Access Layer)**: Този слой управлява физическата връзка между устройствата. Тук се извършват задачи като адресиране на мрежови карти и контрол на достъпа до медията (например Ethernet или Wi-Fi).
- **Приложен слой (Application Layer)**: Тук се намират протоколите, използвани от приложенията за комуникация. Примерите включват:
  - **HTTP** (за уеб трафик),
  - **SMTP** (за електронна поща),
  - **FTP** (за прехвърляне на файлове).
- **TCP/IP в действие:** Когато изпращате съобщение по интернет, то първо се разделя на малки пакети. Всеки пакет съдържа IP заглавие, което включва IP адреса на получателя и изпращача. След това **TCP** контролира предаването на тези пакети, като следи дали всички пакети пристигат правилно и в правилния ред. Ако някой пакет бъде изгубен, **TCP** изисква повторно изпращане.

⇒ **Услуги и сървъри:** Операционната система предлага различни **мрежови услуги** за автоматизиране и улесняване на комуникацията в мрежи. Най-често използваните услуги са:

- **DHCP (Dynamic Host Configuration Protocol)**: Тази услуга автоматично разпределя IP адреси на устройствата в мрежата. Когато ново устройство се свърже към мрежата, DHCP сървърът му присвоява IP адрес, който то използва за комуникация. DHCP също така задава информация за шлюз по подразбиране, DNS сървъри и други мрежови настройки.
- **DNS (Domain Name System)**: DNS преобразува имена на домейни (например www.example.com) в IP адреси, които компютрите могат да използват за комуникация. Без DNS, потребителите ще трябва да въвеждат IP адреси вместо домейн имена, което значително би усложнило употребата на интернет.
- **FTP (File Transfer Protocol)**: FTP е протокол за прехвърляне на файлове между устройства в мрежата. Това е особено полезно за качване и сваляне на големи обеми от данни или за управление на файлове на отдалечени сървъри.

⇒ **Мрежова сигурност:** Операционната система осигурява **сигурност на мрежовите комуникации**, за да защити данните и мрежовите ресурси от неоторизиран достъп и атаки. Мрежовата сигурност включва различни механизми за защита:

- **Защитни стени (Firewalls)**: Защитните стени контролират трафика, който преминава през мрежовия интерфейс на компютъра. Те блокират неоторизиран достъп, като позволяват само разрешени връзки. Защитната стена може да бъде конфигурирана да разрешава или блокира специфични портове, IP адреси или протоколи.
- **SSL/TLS (Secure Sockets Layer/Transport Layer Security)**: Това са криптографски протоколи, които осигуряват сигурна комуникация по

интернет. Те шифроват данните между клиента и сървъра, като гарантират, че информацията не може да бъде прихваната или променена от неоторизирани лица.

- **Механизми за автентикация и авторизация**: ОС предоставя механизми за проверка на самоличността на потребителите и програмите чрез автентикация (като потребителско име и парола) и авторизация (права на достъп до ресурси). Това осигурява контрол кой има достъп до системата и до мрежовите ресурси.

4. **Acts as an Intermediary Between Users and the Hardware (Действа като посредник между потребителите и хардуера):** Една от най-важните функции на операционната система е, че тя действа като посредник между потребителите и хардуера на компютъра. Това означава, че потребителите не взаимодействат директно с хардуера, а вместо това използват операционната система, за да извършват операции като създаване на файлове, отваряне на програми и свързване към интернет.

   - **Потребителски интерфейс**: ОС осигурява графичен интерфейс (GUI), който улеснява взаимодействието на потребителите с компютъра. Това включва прозорци, икони, бутони и менюта.
   - **Команден интерфейс**: В допълнение към GUI, операционните системи предоставят и команден ред (CLI), който дава на напредналите потребители и администратори възможността да изпълняват команди директно за взаимодействие с ОС.
   - **Управление на грешки и сигнали**: ОС обработва грешките в системата и известява потребителите или програмите, ако нещо се обърка (например ако даден файл липсва или ако възникне конфликт на ресурс).

Основни функции на операционната система

1. **Resource Management (Управление на ресурсите): Управлението на ресурсите** е една от основните функции на операционната система (ОС). То включва наблюдение и управление на всички хардуерни и софтуерни ресурси в компютърната система. ОС трябва да разпределя тези ресурси сред процесите и програмите по ефективен начин. Ресурсите могат да включват процесор, памет, дисково пространство и периферни устройства като принтери и скенери.

   - **Процесор**: ОС трябва да определи колко време да даде на всеки процес за използване на процесора (т.нар. процесорно време). Този процес се нарича **планиране на процеси (scheduling)**.
   - **Памет**: ОС управлява RAM паметта, като осигурява достатъчно памет за всяка програма. Това управление се нарича **алокация на паметта**.
   - **Входно/изходни устройства**: ОС контролира достъпа до устройства като клавиатури, мишки, монитори и външни устройства, осигурявайки правилно комуникиране с процесите.

2. **Process Isolation (Изолация на процесите): Изолацията на процесите** е механизъм, чрез който операционната система гарантира, че процесите са независими един от друг и не

могат да взаимодействат директно без позволение. Това осигурява стабилността и сигурността на системата.

- **Какво представлява?** Всеки процес в операционната система има собствено адресно пространство, което е изолирано от това на другите процеси. Това означава, че един процес не може да чете или пише в паметта на друг процес, освен ако ОС не разреши това чрез специални механизми за комуникация между процесите, като **междупроцесна комуникация (Inter-Process Communication, IPC)**.
- **Защо е важно?** Изолацията на процесите предотвратява грешките или зловредните действия в един процес да засегнат останалата част от системата. Ако дадена програма се срине или се държи неправилно, тя няма да може да компрометира други процеси или цялата система.
- **Пример**: Когато използвате уеб браузър и текстов редактор едновременно, изолацията на процесите гарантира, че грешка в уеб браузъра (например срив) няма да засегне текстовия редактор или други работещи програми.

3. **Input/Output Control (Контрол на входно/изходните устройства): Контролът на входно/изходните устройства** е процесът на управление на комуникацията между операционната система и периферните устройства като клавиатури, мишки, принтери, твърди дискове и мрежови интерфейси. Входно/изходните операции са важни, тъй като позволяват взаимодействие между компютъра и външния свят.

   - **Входни устройства**: ОС трябва да управлява сигнали от устройства като клавиатурата и мишката. Тя приема тези сигнали, обработва ги и ги предава на съответните процеси или приложения.
   - **Изходни устройства**: ОС трябва да управлява изхода към устройства като монитори, принтери или мрежови устройства. Например, когато потребителят стартира печат, ОС комуникира с принтера чрез драйверите и осигурява правилната обработка на данните за печат.
   - **Буфериране**: Често операционната система използва буфери, за да съхранява данни преди да ги изпрати на входно/изходно устройство. Това намалява времето за чакане и подобрява ефективността на системата.
   - **Пример**: Когато въвеждате текст с клавиатурата, операционната система улавя всеки натиск на клавиш, обработва го и го изпраща към приложението, което използвате (например текстов редактор). По същия начин, когато изпращате документ към принтер, ОС осигурява правилното предаване на данните към принтера.

4. **Memory Management (Управление на паметта):** ОС управлява оперативната памет, като разпределя необходимото пространство за процесите и освобождава паметта, когато тя вече не е нужна, включително чрез използване на виртуална памет.

5. **File System Management (Управление на файловата система):** ОС организира и управлява файловете и директориите на устройствата за съхранение, като предоставя функции за създаване, четене, писане и изтриване на файлове.

6. **Security and Permissions (Сигурност и права за достъп):** ОС осигурява сигурност чрез контрол на достъпа до ресурси, удостоверяване на потребителите и защита на системата от неоторизиран достъп и зловреден софтуер.

7. **Process and Thread Management (Управление на процесите и нишките):** Процесите са основните единици за изпълнение в операционната система. Нишките (threads) са подкомпоненти на процесите и позволяват едновременната работа на части от един и същ процес. Управлението на процеси и нишки включва създаването, планирането, изпълнението и прекратяването им. ОС контролира как процесите използват процесорния ресурс и кога трябва да бъдат изпълнени.

    - **Пример**: Когато изпълнявате програма като текстов редактор, тя създава процес. В рамките на този процес могат да бъдат създадени множество нишки за обработка на различни задачи, като четене и запис на файлове или обновяване на интерфейса.

User vs Developer

OS Developer point of view (Гледна точка на разработчика на операционната система):

1. **Provides application programming interfaces (API):** От гледна точка на разработчиците, операционната система осигурява **интерфейси за програмиране на приложения (API)**, които представляват набор от функции и методи, чрез които те могат да взаимодействат с операционната система и да използват нейните ресурси. Чрез тези API разработчиците могат да създават софтуер, който да използва системните ресурси без да се налага директно взаимодействие с хардуера.

    ⇒ **Защо е важно за разработчиците?** Чрез API, разработчиците могат да създават приложения, които лесно да се интегрират с операционната система, без да трябва да се занимават с детайлите на хардуера. ОС осигурява абстракция, която позволява на софтуера да бъде преносим и съвместим между различни хардуерни платформи и устройства.

2. **Provides common services:** Операционната система предлага и **общи услуги**, които се използват от различни приложения. Това включва услуги като управление на процеси, памет, сигурност, мрежови връзки и т.н. Тези услуги са достъпни за всички приложения и осигуряват базова инфраструктура, върху която софтуерът може да работи.

    ⇒ **Какво включват общите услуги?**

    - **Управление на процеси**: ОС създава и управлява процесите, като осигурява възможността приложенията да работят паралелно.
    - **Файлова система**: ОС предоставя интерфейс за създаване, четене и писане на файлове и директории.
    - **Управление на мрежи**: Операционната система предоставя стандартни услуги за комуникация по мрежа, като например поддръжка на TCP/IP протоколи за изпращане и получаване на данни.

⇒ **Пример**: Когато разработчик иска да стартира нов процес или да комуникира с друго устройство, операционната система предоставя необходимите услуги и инструменти, за да може това да се случи без необходимостта от ръчно управление на тези ресурси.

OS User point of view (Гледна точка на потребителя на операционната система)

1. **Operating system is there to execute programs (Операционната система е там, за да изпълнява програми):** От гледна точка на потребителя, **основната функция на операционната система** е да стартира и изпълнява програми. Това е най-видимата и важна роля на ОС за обикновения потребител. Потребителите обикновено не се интересуват как ОС управлява хардуера или предоставя API на разработчиците, а единствено какви приложения могат да използват и колко лесно се стартират.

   ⇒ **Пример**: Потребител стартира текстов редактор или уеб браузър. Те не се интересуват как точно операционната система разпределя ресурси или обработва процеси – важно е програмата да се стартира и да работи гладко.

   ⇒ **Защо това е важно за потребителите?** За потребителя, ОС е само средата, която им позволява да използват различни софтуерни приложения, като игри, офис пакети или браузъри. Потребителите очакват ОС да осигури бързото и лесно стартиране на тези програми.

2. **Provides easy interface (Предоставя лесен за използване интерфейс):** ОС предоставя **графичен потребителски интерфейс (GUI)** или команден интерфейс (CLI), които улесняват потребителите в тяхната работа с компютъра. Това включва работен плот, икони, менюта и прозорци, които правят взаимодействието с операционната система и програмите интуитивно и удобно.

   ⇒ **Графичен потребителски интерфейс (GUI)**: Това е интерфейс, който включва визуални елементи като икони, прозорци и бутони, което прави взаимодействието с компютъра лесно за обикновените потребители. Повечето модерни операционни системи като Windows, macOS и Linux предоставят GUI за работа.

   ⇒ **Команден интерфейс (CLI)**: За по-напреднали потребители, ОС предоставя CLI, където командите се въвеждат директно чрез текст. Това е по-мощен, но по-малко интуитивен начин за работа с операционната система.

   ⇒ **Пример**: Потребител използва мишка, за да отвори програма от работния плот, след което прехвърля файл, използвайки лесни за използване икони и менюта.

   ⇒ **Защо това е важно за потребителите?** Потребителите искат да взаимодействат с операционната система по лесен и интуитивен начин. Един добър интерфейс прави компютъра достъпен за хора, които не са техничари, като същевременно предоставя по-сложни инструменти за напредналите потребители.

Hardware Abstraction (Абстракция на хардуера)

Абстракцията на хардуера е един от най-важните аспекти на операционната система. Тя предоставя слой, който скрива сложните детайли на хардуера от приложенията и програмите,

които работят на компютъра. Това означава, че приложенията не трябва да знаят специфичните характеристики на хардуера, за да функционират правилно.

Операционната система служи като посредник между хардуера и софтуера, като осигурява единен интерфейс за достъп до различни хардуерни устройства. Така софтуерът може да работи на различни хардуерни конфигурации, без да се налага да бъде променян или адаптиран за всеки тип устройство.

1. **Can run on different processors (Може да работи на различни процесори, напр. Intel, AMD):** Програмите не се нуждаят от специфично знание за типа процесор, на който работят. Операционната система осигурява абстракция, която позволява на програмите да работят както на **Intel** процесори, така и на **AMD** или други архитектури, без да се налага специално пренаписване или адаптиране.

   ⇒ **Какво означава това за разработчиците?** Те могат да създадат едно приложение, което ще работи на различни процесорни архитектури, без да се тревожат за съвместимост с различните хардуерни конфигурации.

   ⇒ **Как ОС осигурява това?** Операционната система е отговорна за комуникацията с хардуера. Тя превежда общите инструкции на софтуера в специфични команди, които могат да бъдат разбрани и изпълнени от процесора.

2. **No need for separate versions for different processors (Няма нужда от отделни версии за различни процесори):** Благодарение на абстракцията, едно и също приложение може да работи на различни процесори, без да се налага създаване на отделни версии на софтуера. Това улеснява разработчиците и намалява разходите, свързани със създаване и поддържане на множество версии за различни платформи.

   ⇒ **Ползи**: Разработчиците създават единна версия на софтуера, която работи на различни устройства, независимо от хардуерните им специфики.

3. **Works with different storage types and I/O devices transparently (Работи с различни типове съхранение и входно/изходни устройства, напр. HDD, SSD):** Операционната система абстрахира и типовете устройства за съхранение на данни (твърди дискове, SSD, USB памети) и входно/изходните устройства (мишка, клавиатура, принтери и т.н.). За приложенията няма значение дали данните се съхраняват на твърд диск (HDD) или на SSD. Операционната система осигурява единен интерфейс за достъп до данните, независимо от физическия носител.

   ⇒ **Какво означава това за софтуера?** Програмите взаимодействат с файловата система, а не директно с устройството за съхранение. ОС се грижи за това да използва подходящите драйвери за комуникация с устройствата.

   ⇒ **Пример**: Програма за редактиране на документи може да съхрани файл, без да се интересува дали устройството за съхранение е твърд диск или SSD. Операционната система ще се погрижи за това как точно да се запишат данните.

4. **Unaware of how devices are connected (Не знае как са свързани устройствата: USB, мрежа, безжично):** Абстракцията на хардуера означава, че приложенията не трябва да знаят дали дадено устройство е свързано чрез USB, мрежа или безжично (Wi-Fi или

Bluetooth). Операционната система осигурява единен интерфейс за достъп до тези устройства, независимо от начина им на свързване.

⇒ **Пример**: Ако използвате принтер, за приложението няма значение дали принтерът е свързан с USB кабел или по мрежа (LAN или Wi-Fi). Операционната система комуникира с принтера чрез съответните драйвери, без приложението да се занимава с детайлите.

⇒ **Ползи за разработчиците и потребителите**: Приложенията са по-опростени и могат да работят на различни устройства, без да се налага адаптация спрямо начина, по който устройството е свързано.

5. **No need to know network connection type (Няма нужда да знае типа на мрежовата връзка: Wi-Fi, Ethernet):** Операционната система осигурява еднакъв интерфейс за работа с мрежови връзки, независимо дали устройството използва кабелна мрежа (Ethernet) или безжична връзка (Wi-Fi). Програмите използват мрежовите функции на ОС, без да се интересуват как точно данните преминават през мрежата.

⇒ **Пример**: Уеб браузър ще зареди уеб страница, независимо дали връзката с интернет е през Wi-Fi или Ethernet. ОС се грижи за създаването и поддържането на мрежовата връзка.

6. **Multiple applications can share the same hardware simultaneously (Множество приложения могат да използват един и същ хардуер едновременно):** Операционната система управлява многозадачността и осигурява възможността няколко програми да използват едни и същи хардуерни ресурси (например процесор, памет, дискове) едновременно, без да си пречат.

⇒ **Как ОС постига това?** Чрез планиране на процесите и управление на ресурсите, ОС гарантира, че всяка програма получава нужните ресурси, без да пречи на другите процеси. Това включва разпределяне на процесорно време, памет и достъп до входно/изходни устройства.

⇒ **Пример**: Можете да използвате текстов редактор и уеб браузър едновременно, като и двете програми споделят едни и същи ресурси (процесор, памет), а ОС се грижи за управлението на тяхното изпълнение.

Типове операционни системи

1. **Desktop Operating Systems (Операционни системи за настолни компютри):** Това са операционни системи, които обикновено се използват на персонални компютри (PC) и лаптопи. Те предоставят интерфейси и функции, насочени към крайните потребители, които извършват различни задачи като сърфиране в интернет, редактиране на документи, използване на мултимедийни приложения и т.н.

⇒ **Windows**: Най-популярната настолна операционна система, разработена от Microsoft. Тя предлага лесен за използване графичен интерфейс и е съвместима с огромно количество хардуер и софтуер. Основни версии на Windows включват Windows 10 и Windows 11.

⇒ **Linux**: Linux е отворен код операционна система, използвана от много хора по света. Съществуват различни дистрибуции (например Ubuntu, Fedora, Debian), които предлагат различни нива на персонализация и интерфейси. Linux е популярен сред разработчиците и техническите ентусиасти заради своята гъвкавост и сигурност.

⇒ **MacOS**: Операционната система на Apple, предназначена за техните настолни и преносими компютри. macOS е известен със своята интеграция с другите продукти на Apple и с фокуса си върху потребителския интерфейс и мултимедийни приложения.

⇒ **More...**: Има и други по-специализирани или по-малко използвани настолни операционни системи като Chrome OS (на Google, използвана в Chromebook устройства) и FreeBSD.

2. **Mobile Operating Systems (Операционни системи за мобилни устройства):** Мобилните операционни системи са създадени специално за използване на мобилни устройства като смартфони и таблети. Те са оптимизирани за работа с мобилни процесори, батерии и сензорни екрани.

   ⇒ **Android**: Най-разпространената операционна система за смартфони и таблети, разработена от Google. Android е базиран на Linux и поддържа голямо разнообразие от приложения чрез Google Play Store. Android е гъвкав и може да се използва на устройства с различни хардуерни конфигурации.

   ⇒ **iOS**: Операционната система на Apple за техните мобилни устройства, като iPhone и iPad. iOS е затворена платформа, която е силно интегрирана с екосистемата на Apple и предлага стабилност и високо ниво на сигурност.

   ⇒ **More...**: Други мобилни операционни системи включват Windows Phone (вече прекратена), както и специализирани версии на Linux като Sailfish OS.

3. **Server Operating Systems (Операционни системи за сървъри):** Сървърните операционни системи са проектирани да обслужват множество клиенти и да управляват ресурси като бази данни, уебсайтове и мрежови услуги. Те осигуряват стабилност, сигурност и възможност за работа с много потребители едновременно.

   ⇒ **Linux**: Linux е изключително популярен в света на сървърните операционни системи, благодарение на своята гъвкавост, сигурност и отворен код. Сървърни дистрибуции като Ubuntu Server, CentOS и Red Hat са широко използвани за управление на уебсървъри, бази данни и други корпоративни приложения.

   ⇒ **Windows Server**: Windows също има версия за сървъри, наречена **Windows Server**, която се използва в много корпоративни среди. Тя предлага възможности за управление на Active Directory, виртуализация с Hyper-V, както и интеграция с други продукти на Microsoft като SQL Server и Azure.

   ⇒ **Unix**: Unix-базирани системи като BSD (Berkeley Software Distribution), AIX (на IBM) и HPUX (на Hewlett-Packard) също се използват за сървъри, особено в големи корпорации и университетски среди. Те са известни с високата си надеждност и стабилност.

   ⇒ **More...**: Има и други сървърни операционни системи като Solaris (на Oracle), използвани за специфични корпоративни и индустриални приложения.

4. **Embedded Operating Systems (Вградени операционни системи):** Тези операционни системи са предназначени за **вградени системи**, които обикновено изпълняват една или малко на брой функции. Такива системи могат да бъдат открити в домакински уреди, автомобили, индустриални машини и други устройства с ограничени ресурси.

   ⇒ **Linux**: Linux е широко използван във вградени системи, благодарение на своята мащабируемост и възможност за модификация. Специализирани версии на Linux като **Embedded Linux** се използват в устройства като рутери, телевизори и автомобили.

   ⇒ **Minix**: Minix е малка, бърза и сигурна операционна система, използвана главно за образователни цели, но също така намира приложение във вградени системи.

   ⇒ **Windows CE**: Това е олекотена версия на Windows, разработена специално за вградени системи. Тя е използвана в различни индустриални и потребителски устройства.

   ⇒ **More...**: Други операционни системи за вградени системи включват VxWorks и Contik

5. **Real-Time Operating Systems (RTOS) – Операционни системи в реално време:** RTOS са специализирани операционни системи, предназначени да осигуряват **определени задачи в точно определено време**. Те се използват в системи, където времето за реакция е критично, като медицински устройства, автомобили, космически апарати и индустриални системи за управление.

   ⇒ **QNX**: Това е мощна RTOS, използвана в автомобилостроенето, медицинските устройства и телекомуникациите. Тя осигурява стабилност и бързина при критично важни задачи.

   ⇒ **RTLinux**: Това е специална версия на Linux, проектирана за работа в реално време. Тя позволява изпълнение на критични задачи с точно определени времеви ограничения.

   ⇒ **Windows CE**: Въпреки че Windows CE често се използва във вградени системи, тя също има версия за реално време, която намира приложение в индустриалните контролни системи.

   ⇒ **More...**: Други RTOS включват VxWorks и FreeRTOS, използвани в широк спектър от индустриални приложения и IoT устройства.

Архитектура на операционните системи

Операционната система е разделена на две основни области: **пространството на потребителя (User Space)** и **пространството на ядрото (Kernel Space)**. Тези две зони осигуряват контролирана комуникация между приложенията и хардуера на компютъра.

1. **User Space (Потребителско пространство):** Това е зоната, в която работят приложенията на потребителя. Програмите нямат пряк достъп до хардуера или критичните функции на операционната система, а комуникират с ядрото чрез специални повиквания, наречени **системни повиквания (syscalls)**.

- **Потребителски приложения (User Applications)**: Това са програмите, които потребителите стартират, като текстови редактори, браузъри, игри и т.н. Тези приложения изпълняват задачи от името на потребителя, но нямат директен достъп до хардуера.
- **Операционни системни услуги (Operating System Services)**: Това са услуги, които операционната система предоставя на потребителските приложения, като управление на файлове, мрежова комуникация и др.
- **Основни библиотеки (Core Libraries)**: Това са библиотеки, които предоставят интерфейси за програмите и улесняват използването на системните ресурси. Например, стандартни библиотеки като glibc в Linux или msvcrt в Windows съдържат често използвани функции за достъп до операционната система.

2. **Kernel Space (Пространство на ядрото):** Пространството на ядрото е сърцето на операционната система. То осигурява директен достъп до хардуера и контролира всички системни ресурси. Това е защитена зона, до която потребителските приложения нямат достъп директно, за да се гарантира стабилността и сигурността на системата.

   - **Ядро на операционната система (OS Kernel)**: Ядрото е основният компонент на операционната система, който управлява взаимодействието между хардуера и софтуера. В ядрото се намират следните основни модули:
     - **Мениджър на паметта (Memory Manager)**: Управлява разпределението на оперативната памет (RAM) между различните процеси и осигурява виртуална памет, когато физическата памет е недостатъчна.
     - **Мениджър на процесите (Process Scheduler)**: Контролира разпределението на процесорния ресурс между различните процеси, като решава кой процес да бъде изпълнен и кога.
     - **Мениджър на входно/изходни устройства (I/O Manager)**: Управлява взаимодействието с входно/изходните устройства като клавиатура, мишка, дискове и др.
     - **Мениджър на мрежата (Network Manager)**: Контролира мрежовата комуникация и предоставя услуги за изпращане и получаване на данни по мрежата.
   - **Модули на ядрото и драйвери (Kernel Modules / Drivers)**: Това са модули и драйвери, които осигуряват взаимодействие с различни хардуерни устройства. Те се зареждат от ядрото и предоставят интерфейс за управление на устройства като видеокарти, мрежови адаптери, устройства за съхранение и USB устройства.
     - **Драйвери за файлови системи (Filesystems)**: Контролират как се съхраняват и достъпват файловете на дисковете.
     - **Видео устройства (Video Devices)**: Драйвери, които управляват графичните адаптери и дисплеите.
     - **Устройства за въвеждане (Keyboard and Mouse)**: Контролират входящите сигнали от клавиатури, мишки и други входни устройства.
     - **Драйвери за устройства за съхранение (Storage Devices)**: Тези драйвери управляват твърди дискове, SSD, и други устройства за съхранение на данни.
     - **Мрежови устройства (Network Devices)**: Драйверите, които управляват мрежовите карти и осигуряват комуникация по интернет или локална мрежа.

- **USB**: Управлява всички устройства, свързани чрез USB портове.

3. **Взаимодействие между потребителското пространство и ядрото:** Приложенията в **User Space** не могат директно да комуникират с хардуера или да използват системните ресурси без посредничеството на ядрото. За да получат достъп до системни ресурси, те правят заявки чрез **системни повиквания (syscalls)**. Ядрото получава тези заявки и ги обработва, осигурявайки необходимите ресурси или извършвайки заявените операции.

   - **Системни повиквания (Syscalls)**: Това са интерфейсите, чрез които приложенията искат достъп до функции на операционната система. Например, когато приложение иска да отвори файл или да стартира нов процес, то използва системно повикване. Ядрото обработва това повикване и връща резултата обратно към приложението.

Сигнали (Signals): Операционната система може да изпраща **сигнали** към приложенията, за да ги уведомява за определени събития, като грешки, изтичане на време, спиране или завършване на процес. Това е начинът, по който ОС може да комуникира с приложенията и да им предава информация относно техния статус или изпълнение.

Какво е ядро (Kernel)

Ядрото е основният компонент на операционната система и представлява сърцето на всяка съвременна ОС. То отговаря за комуникацията между хардуера и софтуера, като осигурява контрол върху всички ресурси на системата.

1. **The core component of the operating system (Основният компонент на операционната система):** Ядрото е най-важната част от операционната система, без която тя не може да функционира. То осъществява всички основни задачи по управлението на хардуера, като обработка на процеси, управление на паметта, комуникация с входно/изходните устройства и осигуряване на сигурност. Ядрото гарантира, че различните програми и процеси могат да работят заедно и да използват хардуерните ресурси по контролиран начин.

2. **Acts as a bridge between software applications and the hardware of a computer (Действа като мост между софтуерните приложения и хардуера на компютъра):** Основната роля на ядрото е да действа като посредник между софтуера и хардуера. Програмите не комуникират директно с хардуера, а използват ядрото, което осигурява безопасен и контролиран достъп до хардуерните компоненти като процесор, памет, мрежови устройства и др.

   - **Пример**: Когато дадена програма иска да записва файл на твърдия диск, тя прави заявка към ядрото чрез системно повикване. Ядрото обработва тази заявка, като комуникира с устройството за съхранение и извършва необходимите действия.

3. **Provides essential services for applications (Предоставя основни услуги за приложенията):** Ядрото предоставя важни услуги за работата на приложенията, като управление на процесите, разпределение на паметта, управление на файловата система, работа с мрежови връзки и входно/изходни устройства. Тези услуги са от съществено значение за правилното функциониране на софтуера и позволяват на програмите да използват ресурсите на системата безпроблемно.

- **Услуги на ядрото**:
    - **Планиране на процеси**: Разпределя процесорното време между различните задачи и процеси.
    - **Управление на паметта**: Осигурява необходимата оперативна памет на програмите.
    - **Управление на файлове**: Позволява на програмите да четат, записват и модифицират файлове на устройството за съхранение.
    - **Мрежова комуникация**: Позволява на програмите да комуникират през интернет или локални мрежи.

4. **Provides abstraction of hardware components (Осигурява абстракция на хардуерните компоненти):** Ядрото абстрахира сложността на хардуера от софтуера. Това означава, че приложенията не трябва да знаят детайлите на конкретния хардуер, с който работят. Те използват системни повиквания и API, за да комуникират с ядрото, което от своя страна се грижи за взаимодействието с хардуерните устройства.

    - **Пример**: Независимо дали компютърът има SSD или HDD, програмата, която записва файл, няма нужда да знае този детайл. Ядрото се грижи за комуникацията с устройството за съхранение и осигурява унифициран интерфейс за работа.

Разликите между монолитно ядро, микроядро и хибридно ядро

1. **Monolithic Kernel (Монолитно ядро):** Монолитното ядро е тип ядро, при което всички основни функции на операционната система се изпълняват в едно цяло, в едно и също адресно пространство. Това включва управление на процеси, памет, файлове, устройства и др. Всички тези компоненти работят в **Kernel Space** (пространството на ядрото), което увеличава ефективността, тъй като комуникацията между тези функции е много бърза.

    - **Основни характеристики**:
        - **Всички основни функции в едно пространство**: Управлението на процеси, памет, файлове и устройства се извършва в едно цяло и на едно място.
        - **Висока ефективност**: Тъй като няма нужда от преминаване между различни адресни пространства, изпълнението на задачи е много бързо.
        - **По-трудна поддръжка**: Поради сложността на монолитните ядра, те могат да бъдат по-трудни за поддръжка и обновяване, тъй като всяка промяна в една част може да засегне всички останали.
    - **Пример**: **Linux** и **Unix** са примери за операционни системи, които използват монолитни ядра. Те имат висока производителност, но промени в ядрото изискват прекомпилиране на цялото ядро.

2. **Microkernel (Микроядро):** Микроядрото е опростена версия на ядрото, което минимизира функциите, които се изпълняват в **Kernel Space**. Самото микроядро изпълнява само най-важните задачи, като управление на процеси, памет и комуникация между различните компоненти. Останалите функции като управление на файловата система и устройствата се изпълняват извън ядрото, в **User Space**, като отделни модули.

- **Основни характеристики**:
  - **Минимални функции в ядрото**: Основните функции са сведени до минимум – обикновено само управление на процесите, паметта и съобщенията между процесите.
  - **Делегиране на задачи на драйвери**: Останалите функции като файлови системи и управление на устройства се изпълняват от драйвери и модули, които работят в потребителското пространство.
  - **По-голяма сигурност и стабилност**: Тъй като по-малко функции работят в ядрото, възможността за сривове или грешки е по-малка. Освен това, дефектите в драйверите не засягат цялата система, защото те работят извън ядрото.
- **Пример**: **Minix** и **QNX** са примери за операционни системи, които използват микроядра. Микроядрата имат по-голяма модулност и сигурност, но за сметка на това комуникацията между модулите може да бъде по-бавна.

3. **Hybrid Kernel (Хибридно ядро):** Хибридното ядро комбинира аспекти от както монолитните, така и микроядрата, за да постигне баланс между ефективност и модулност. При хибридните ядра някои компоненти на системата работят в ядрото за по-добра производителност, а други в потребителското пространство, за по-голяма сигурност и гъвкавост.

- **Основни характеристики**:
  - **Комбинира характеристики на монолитните и микроядрата**: Част от функциите на ядрото остават в Kernel Space за по-бързо изпълнение, докато други функции (като драйверите) работят в User Space.
  - **Опит за баланс между производителност и модулност**: Хибридните ядра целят да предоставят висока производителност, като същевременно позволяват модулност и сигурност.
  - **По-гъвкава структура**: Лесно може да се добавят нови функционалности чрез модули, без да е необходимо прекомпилиране на цялото ядро.
- **Пример**: **Windows NT** и **macOS** са примери за операционни системи с хибридни ядра. Те комбинират предимствата на монолитните и микроядрата, като предлагат производителност и стабилност.

Какво е разделяне на дискове (Partitioning)

Разделянето на дискове е процесът на разделяне на физически твърд диск на няколко логически секции, наречени **дялове (partitions)**. Всеки дял може да функционира като отделен логически диск в системата, въпреки че всички дялове се намират на един и същи физически диск.

- **Какво представлява?**: Това е методът за създаване на независими пространства в един физически диск. Тези пространства могат да се използват за различни цели, като например операционни системи, данни или временни файлове.

⇒ **Защо се използва разделяне (Why Partition)?** - Разделянето на диск може да донесе няколко предимства за управление на данни, производителност и сигурност:

1. **Организиране на данни и разделяне за различни цели:** Разделянето на диск позволява да се създадат различни **дялове за различни цели**, като:

   - **Операционна система (OS)**: Един дял може да се използва за инсталиране на операционната система. Това позволява лесна преинсталация или обновяване на ОС, без да се засягат данните на другите дялове.
   - **Съхранение на данни (Data Storage)**: Втори дял може да се използва за съхранение на лични данни, като документи, снимки и видео, отделно от операционната система.
   - **Временни файлове (Temporary Storage)**: Може да се създаде дял за временни файлове или файлове на подменяне (swap), за да се подобри управлението на дисковите ресурси.
   - **Пример**: Ако имате един дял за операционната система и отделен дял за данни, можете да преинсталирате ОС без да загубите личните си файлове, които се намират на другия дял.

2. **Подобряване на управлението на данни и производителността:** Разделянето на диск може да подобри управлението на данни, като предоставя отделни пространства за различни видове файлове. Например:

   - Лесно се организират данни по категории, като системни файлове, лични данни и временни файлове, което намалява риска от объркване или загуба на данни.
   - Разделените дялове могат да подобрят производителността на системата, като се използват за специфични задачи (например swap дял за виртуална памет).
   - **Пример**: Ако операционната система и данните са на различни дялове, файловете на ОС няма да се смесват с личните файлове, което улеснява управлението на системата.

3. **Различни файлови системи за различни дялове:** Всеки дял на диска може да бъде форматиран с **различна файлова система**, което позволява по-гъвкаво управление на данните в зависимост от нуждите:

   - Един дял може да бъде форматиран с NTFS за Windows, докато друг може да използва EXT4 за Linux, което прави възможно инсталирането на различни операционни системи на един диск.
   - Някои дялове могат да използват специализирани файлови системи за конкретни задачи, като FAT32 за съвместимост с по-стари системи или exFAT за големи файлове.
   - **Пример**: Ако искате да инсталирате както Windows, така и Linux на един компютър (dual boot), можете да разделите диска и да използвате различни файлови системи за всеки от дяловете.

Файловата система

1. **Какво е файлова система? - Файловата система** е методът, чрез който данните се **съхраняват, организират и извличат** от устройство за съхранение (твърд диск, SSD, USB и др.). Тя определя как файловете се съхраняват на диска, как се намират и как се организират, за да могат потребителите и приложенията да имат достъп до тях по ефективен начин.

   - **Основна роля**: Файловата система създава структура върху устройството за съхранение, която позволява данните да бъдат организирани в директории и файлове, което прави управлението на данните лесно и структурирано.

Функции на файловата система: Файловата система изпълнява няколко важни функции, които улесняват съхранението и достъпа до данни на компютъра.

1. **Организация и управление на файлове:** Файловата система създава логическа структура върху физическия диск, която позволява данните да бъдат съхранявани във формата на **файлове и директории (папки)**. Тази структура улеснява организирането и достъпа до данните.

   - **Как се организира?**: Файловете се съхраняват в йерархична структура, където всяка директория може да съдържа файлове и други поддиректории. Това позволява лесно създаване, преместване и изтриване на файлове.
   - **Пример**: На един диск може да имате директория "Документи", която съдържа различни файлове, както и поддиректории за специфични проекти или категории документи.

2. **Контрол на достъпа и права (Access Control and Permissions):** Файловата система предоставя механизми за управление на правата за достъп до файловете и директориите. Това позволява задаването на права за различни потребители или групи.

   - **Примери за права**:
     - **Четене (Read)**: Потребителят има право да чете съдържанието на файла.
     - **Записване (Write)**: Потребителят може да редактира съдържанието на файла или директорията.
     - **Изпълнение (Execute)**: Потребителят може да стартира програмни файлове или скриптове.
   - **Защо е важно?**: Този контрол гарантира, че само оторизирани потребители имат достъп до определени файлове и че критични системни файлове не могат да бъдат променяни от обикновени потребители.

3. **Поддръжка на метаданни за файловете (Metadata for Files):** Файловата система съхранява **метаданни** за всеки файл. Това са допълнителни данни, които описват самия файл, без да включват съдържанието му.

- **Примери за метаданни**:
    - **Времеви отпечатъци (Timestamps)**: Кога файлът е създаден, последно променен или достъпен.
    - **Размер на файла (File Size)**: Колко място заема файлът на диска.
    - **Собственик и група**: Кой е собственикът на файла и към коя група потребители принадлежи.
    - **Тип на файла**: Дали файлът е обикновен текстов файл, директория или изпълним файл.
- **Защо е важно?**: Метаданните позволяват ефективно управление и индексиране на файловете. Също така, те са полезни за сигурността и контрола на достъпа, тъй като съдържат информация за правата върху файла.

Основните понятия за файлове и директории в контекста на операционната система и файловата система

1. **Какво е файл? - Файлът** е логическа групировка от свързани данни, съхранени на физически носител (например твърд диск, SSD, USB устройство). Всеки файл съдържа определено количество информация, която може да бъде текст, изображения, видео, аудио, програмен код или всякакви други данни.

    - **Идентифициране чрез име на файла (Filename)**: Всеки файл се идентифицира с уникално име, което позволява на операционната система и потребителите да го намират и използват. Името на файла обикновено включва и **разширение** (напр. .txt, .jpg, .exe), което указва типа на файла.
    - **Пример**: Файлът document.txt е текстов файл, който съдържа текстова информация. Името document идентифицира файла, а разширението .txt указва, че това е текстов файл.

2. **Какво е директория? - Директорията** (известна също като **папка**) е йерархична колекция от файлове и други директории. Тя позволява организацията на файловете в логически структури, което улеснява тяхното управление и намиране.

    - **Йерархична структура**: Директориите са структурирани в йерархия, където могат да съдържат както файлове, така и други поддиректории (папки). Това създава дървовидна структура, която позволява лесно подреждане и навигация между различни файлове и папки.
    - **Пример**: Директорията Документи може да съдържа файлове като project.docx и поддиректория Работа, която от своя страна съдържа други файлове и поддиректории.

Процес

Какво е процес? - Процесът е изпълняващ се екземпляр на програма. Когато програма се стартира, операционната система създава процес, който включва всички необходими ресурси за изпълнението ѝ.

- **Екземпляр на работеща програма**: Всеки път, когато дадена програма се стартира, тя се превръща в процес. Програмата сама по себе си е просто набор от инструкции, но когато започне да се изпълнява, операционната система я превръща в процес, който включва инструкции, данни, файлове и ресурси, нужни за работата ѝ.
- **ОС създава процес за стартиране на програма**: Операционната система е отговорна за създаването на процеси и управлението им. Това включва разпределяне на процесорно време, памет и други ресурси на процесите. Когато стартирате програма (например текстов редактор), ОС създава нов процес и предоставя нужните ресурси за неговото изпълнение.
- **Стартирането на приложение създава процес**: Всеки път, когато стартирате ново приложение на вашия компютър, като например уеб браузър или игра, операционната система създава процес, който представлява тази програма. Ако стартирате същата програма няколко пъти, ОС ще създаде отделен процес за всеки екземпляр.

Междупроцесната комуникация (Inter-Process Communication, IPC)

1. **Какво е IPC (Inter-Process Communication)? - Междупроцесната комуникация (IPC)** е механизъм, който позволява на различни процеси в операционната система да обменят данни и да си сътрудничат. Тъй като процесите в съвременните операционни системи са изолирани един от друг (за да се гарантира стабилност и сигурност), IPC осигурява начин за комуникация между тях, така че да могат да координират своите действия и да споделят информация.

2. **Основни функции на IPC:**

- **Механизъм за обмен на данни:** IPC осигурява на процесите средство за изпращане и получаване на данни помежду си. Без IPC, всеки процес би бил изолиран и неспособен да комуникира с другите процеси в системата, което би ограничило възможностите за изпълнение на по-сложни задачи.
    - **Пример**: В система с няколко активни процеса, уеб браузърът може да използва един процес за основния интерфейс и друг процес за обработка на видео съдържание. Тези два процеса могат да комуникират помежду си чрез IPC, за да синхронизират действията си.
- **Координация между процеси за изпълнение на сложни задачи:** IPC позволява координация между различни процеси за изпълнение на задачи, които не могат да бъдат изпълнени от един процес самостоятелно. Когато различни процеси работят заедно по обща цел, те трябва да могат да комуникират ефективно и да споделят ресурси и данни.

- **Пример**: Когато дадена програма използва многопоточна или многопроцесорна архитектура, тя разделя задачите между множество процеси. IPC позволява тези процеси да координират действията си, за да завършат задачата заедно.

3. **Защо е важно IPC? -** IPC е критично важно за ефективната работа на операционните системи и за взаимодействието между процесите. Без IPC много приложения не биха могли да изпълняват задачи, които изискват взаимодействие между различни компоненти.

⇒ **Споделяне на ресурси (Resource Sharing) -** IPC осигурява механизми за **споделяне на ресурси** между процеси, като памет, файлове и други системни ресурси. Например, два или повече процеса може да трябва да използват едни и същи данни, които се съхраняват в обща памет или файл.

- **Пример**: В операционна система с виртуална памет, процесите могат да споделят определени сегменти от паметта чрез механизми като **споделена памет (shared memory)**, където всички процеси имат достъп до едни и същи данни.

⇒ **Синхронизация на процесите (Process Synchronization):** IPC осигурява средства за **синхронизация** между процесите, като гарантира, че действията на един процес не пречат на друг процес. Това е особено важно при изпълнението на паралелни задачи, където процесите трябва да спазват определен ред на изпълнение или да се избягват конфликти при достъп до споделени ресурси.

- **Пример**: Ако два процеса трябва да пишат в един и същ файл, IPC може да се използва, за да гарантира, че само един процес пише в даден момент, като по този начин се избягва повреда на файла или загуба на данни. За тази цел могат да се използват **семафори** или **мютекси** (mutexes) за контролиране на достъпа.

⇒ **Модулност и ефективност (Modularity and Efficiency):** IPC насърчава **модулността** на системата, което означава, че големите задачи могат да бъдат разделени на по-малки, специализирани процеси, които комуникират и си сътрудничат чрез IPC. Това подобрява **ефективността**, тъй като разделените модули могат да бъдат изпълнявани паралелно и да обменят данни помежду си.

- **Пример**: Модерните операционни системи използват микроядра, където основните функции на ОС са разделени на малки модули, които комуникират чрез IPC. Това прави системата по-устойчива на грешки, тъй като проблем в един модул не се разпространява към останалите.

4. **Видове механизми за IPC:**

⇒ **Споделена памет (Shared Memory)**: Процесите могат да споделят област от паметта, която може да бъде достъпна от всички участващи процеси. Това е един от най-бързите методи за IPC, но изисква синхронизация, за да се избегнат конфликти.

⇒ **Комуникация чрез съобщения (Message Passing)**: Процесите могат да изпращат и получават съобщения помежду си чрез пощенски кутии или канали за съобщения. Този метод е по-бавен от споделената памет, но е по-безопасен, тъй като всяко съобщение е изолирано.

⇒ **Семафори (Semaphores)**: Семафорите се използват за синхронизация между процесите и за контрол на достъпа до споделени ресурси.

⇒ **Тръби (Pipes)**: Това е метод за предаване на данни между процеси в линейна последователност, като еднопосочен поток от данни.

Механизми за междупроцесна комуникация (IPC)

1. Pipes (Тръби): **Тръбите** са един от най-старите и най-простите механизми за междупроцесна комуникация. Те позволяват на два процеса да комуникират помежду си чрез изпращане и получаване на данни в линеен поток. Тръбите са еднопосочни, което означава, че данните могат да текат само в една посока: от един процес към друг.

   ⇒ **Как работят?**: Единият процес записва данни в тръбата, а другият процес ги чете. След като данните бъдат прочетени, те не могат да бъдат възстановени.

   ⇒ **Пример**: Процес А създава тръба и изпраща данни чрез нея, а процес Б приема тези данни. Този тип комуникация често се използва за връзка между родителски и дъщерни процеси.

   ⇒ **Ограничения**: Тъй като тръбите са еднопосочни, ако е необходимо двупосочно предаване на данни, трябва да се създадат две тръби.

2. Message Queues (Опашки за съобщения): **Опашките за съобщения** са механизъм, при който процесите могат да изпращат и получават съобщения в асинхронен режим. За разлика от тръбите, опашките позволяват на няколко процеса да комуникират чрез изпращане на съобщения в дадена опашка, която се управлява от операционната система.

   ⇒ **Как работят?**: Съобщенията се поставят в опашка и чакат да бъдат прочетени от друг процес. Съобщенията могат да имат приоритет, което означава, че някои съобщения могат да бъдат прочетени преди други, в зависимост от тяхната важност.

   ⇒ **Предимства**: Опашките за съобщения са добър начин за организиране на съобщенията между процеси, без да е необходимо процесите да работят едновременно. Един процес може да изпрати съобщение и да продължи с изпълнението си, докато другият процес го получи по-късно.

   ⇒ **Пример**: Процес А поставя заявка в опашката за съобщения, а процес Б я прочита, когато е готов да я обработи. Това се използва в системи, които изискват асинхронна комуникация.

3. Shared Memory (Споделена памет): **Споделената памет** е един от най-бързите механизми за междупроцесна комуникация, тъй като позволява на два или повече процеса да имат достъп до една и съща област от паметта. Процесите могат директно да четат и записват данни в споделената памет.

   ⇒ **Как работи?**: Операционната система създава сегмент от памет, който може да бъде достъпен от всички участващи процеси. Тези процеси могат да четат и пишат в този

сегмент, като се синхронизират чрез допълнителни механизми като семафори или мютекси, за да се избегне конфликт.

⇒ **Предимства**: Споделената памет осигурява изключително бърза комуникация, защото не изисква системни повиквания за четене и запис на данни. Всички процеси могат да достъпват данните директно в RAM.

⇒ **Пример**: Процес А и процес Б споделят сегмент от паметта, в който пишат и четат данни. Те използват мютекс, за да синхронизират достъпа и да избегнат едновременен запис в паметта.

⇒ **Ограничения**: Изисква допълнителни механизми за синхронизация, за да се предотвратят конфликти между процесите.

4. Semaphores/Mutex (Семафори и Мютекси): **Семафорите** и **мютексите** са механизми за **синхронизация** между процесите, които осигуряват безопасен достъп до споделени ресурси. Те предотвратяват едновременен достъп на няколко процеса до един и същ ресурс, което би могло да доведе до повреда на данни или нестабилност.

⇒ **Семафори**: Това са променливи, които се използват за управление на достъпа до споделен ресурс, като позволяват ограничен брой процеси да имат достъп едновременно. Те могат да бъдат **бинарни** (разрешават само един процес) или **броячни** (разрешават определен брой процеси).

⇒ **Мютекси**: Мютексите са подобни на бинарните семафори, но осигуряват изключителен достъп до ресурса. Само един процес може да "заключи" мютекса и да получи достъп до ресурса.

⇒ **Пример**: Ако два процеса искат да пишат в един и същ файл, мютекс или семафор може да гарантира, че само един процес ще получи достъп до файла по едно и също време.

5. Sockets (Сокети): **Сокетите** осигуряват механизъм за комуникация между процеси, които могат да бъдат на различни машини или в различни мрежи. Те се използват основно за мрежова комуникация, като осигуряват интерфейс за изпращане и получаване на данни по TCP/IP или други мрежови протоколи.

⇒ **Как работят?**: Сокетите създават връзка между два процеса чрез IP адреси и портове. Един процес може да изпрати данни през мрежата до друг процес, който приема тези данни чрез свързан сокет.

⇒ **Предимства**: Сокетите са изключително гъвкави и позволяват комуникация между процеси, които не се намират на една и съща машина.

⇒ **Пример**: Когато браузърът изпраща заявка до уеб сървър, той използва сокети за комуникация по TCP/IP, за да изпрати заявката и да получи отговор.

6. Unix Sockets (Unix Сокети): **Unix сокетите** са специален тип сокети, които се използват за комуникация между процеси на една и съща машина, без да се използва мрежова комуникация. Те се считат за локални сокети и са по-ефективни за комуникация в рамките на една операционна система, тъй като избягват мрежовия стек.

   ⇒ **Как работят?**: Unix сокетите работят подобно на мрежовите сокети, но вместо да използват IP адреси, те използват пътища в файловата система за идентифициране на крайни точки.

   ⇒ **Предимства**: Unix сокетите са по-бързи и по-ефективни за локална комуникация между процеси, тъй като не преминават през мрежовия стек. Те са идеални за IPC в рамките на една машина.

   ⇒ **Пример**: Въпреки че браузър и уеб сървър на една и съща машина биха могли да комуникират чрез TCP сокети, Unix сокетите биха били по-бърз и по-ефективен избор.

Какво е Daemon или Service?

Демон (Daemon) е дългоработещ процес, който се изпълнява във фонов режим и осигурява определена услуга или функционалност, която не изисква директно взаимодействие с потребителя. Демоните обикновено са конфигурирани да се стартират заедно с операционната система и продължават да работят във фонов режим, докато системата е включена.

- **Работи във фонов режим**: За разлика от стандартните приложения, които потребителите стартират и взаимодействат директно с тях, демоните работят постоянно във фонов режим, без да се налага потребителят да ги стартира или управлява ръчно.
- **Специфична функция без намеса на потребителя**: Демоните изпълняват специфични задачи, които са важни за работата на системата или приложенията, като осигуряват услуги като уеб сървър, база данни, наблюдение на системата и др.
- **Конфигуриране за автоматично стартиране с операционната система**: Повечето демони се конфигурират да се стартират автоматично, когато операционната система се зарежда, за да гарантират, че услугите, които предоставят, са налични веднага след стартиране на системата.
- **Демон срещу Услуга (Service)**: В Unix-подобни операционни системи (като Linux и macOS) тези процеси се наричат **демони**, докато в Windows системите подобни процеси се наричат **услуги (services)**.

⇒ **Основни характеристики на демоните:**

1. **Дългоработещ процес**: Демоните работят дълго време, често от стартирането на системата до нейното изключване. Те не приключват, след като изпълнят задача, а остават активни, за да отговорят на бъдещи заявки или да изпълняват повтарящи се задачи.
2. **Автоматично стартиране**: Демоните често са конфигурирани да се стартират автоматично с операционната система, без намеса от потребителя. Това е важно за

услуги, от които системата зависи за нормалното си функциониране, като мрежови услуги, бази данни и т.н.

3. **Без потребителско взаимодействие**: Демоните изпълняват своите задачи във фонов режим и не се нуждаят от взаимодействие с потребителя. Вместо това те чакат събития, заявки или сигнали, за да изпълнят своята функция.
4. **Модулност и специализация**: Всеки демон има специфична цел и обикновено изпълнява една или няколко специализирани функции. Например, уеб сървърът обработва HTTP заявки, докато базата данни съхранява и извлича данни.

⇒ **Примери за демони:**

1. **Уеб сървър (Apache, Nginx, IIS)**:
   - **Apache и Nginx**: Това са два популярни уеб сървъра, които функционират като демони. Те обработват HTTP заявки и доставят уеб страници на потребителите. След като бъдат стартирани, те работят във фонов режим и чакат заявки от клиенти.
   - **IIS (Internet Information Services)**: Това е уеб сървърът на Microsoft за Windows. Както Apache и Nginx, той също работи като услуга в Windows и осигурява уеб услуги на потребителите.
2. **Сървъри за бази данни (MySQL, MongoDB, MSSQL)**:
   - **MySQL и MongoDB**: Това са бази данни, които работят като демони. Те се грижат за съхраняването и управлението на големи обеми от данни. След като бъдат стартирани, те чакат заявки от приложения или потребители, които искат да съхраняват или извличат данни.
   - **MSSQL (Microsoft SQL Server)**: Това е сървър за база данни на Microsoft, който работи като услуга в Windows и предоставя същите функции като MySQL и MongoDB.
3. **Други примери**:
   - **Cron**: В Unix системите демонът **cron** се използва за изпълнение на автоматизирани задачи (jobs) в определени времеви интервали. Той периодично проверява конфигурационни файлове и изпълнява задачи като архивиране на файлове, обновяване на системата и други автоматизирани задачи.
   - **SSHD (Secure Shell Daemon)**: Това е демон, който управлява **SSH връзките** в Unix-базирани системи. Той позволява отдалечен достъп и управление на системата чрез криптирани връзки.

⇒ **Защо демоните са важни?**

- **Автоматизация на задачи**: Демоните улесняват автоматизацията на много системни задачи, като например архивиране на данни, мониторинг на мрежата, управление на услуги и други.
- **Непрекъснато предоставяне на услуги**: Много от основните услуги на системата се предоставят чрез демони, които работят непрекъснато и осигуряват стабилност и наличност на тези услуги (напр. бази данни, уеб сървъри и др.).

- **Модулност и гъвкавост**: Демоните разделят функциите на системата на отделни модули, които могат да се стартират и управляват поотделно, което улеснява поддръжката и управлението на системата.

Многозадачност (Multitasking)

1. **CPU Core == Single Task (Едно ядро на процесора == Една задача):** Един **ядро на процесора (CPU core)** може да изпълнява само **една задача** в даден момент. Това означава, че процесорът може да изпълнява инструкциите на само един процес наведнъж. Ако системата има само едно ядро, то на практика може да се изпълнява само един процес в даден момент.
    - **Какво се случва?**: Процесорът може да обработва само един поток от инструкции наведнъж за всяко ядро. Това е ограничение на физическия хардуер на процесора.

2. **Какво е многозадачност (What is Multitasking)? - Многозадачността** е методът, при който операционната система позволява на **няколко процеса да споделят един и същи процесор (CPU) и други системни ресурси**. Това създава впечатлението, че множество задачи се изпълняват едновременно, въпреки че в действителност процесорът превключва бързо между различните задачи.

    - **Споделяне на CPU**: Многозадачността разчита на **разпределянето на процесорното време** между различни процеси. Процесорът бързо превключва между различни задачи, така че на потребителите и приложенията им изглежда, че всички процеси се изпълняват едновременно.
    - **Как работи?**: Операционната система планира процесите, като им разпределя "времеви дялове" (time slices), в които процесорът обработва техните инструкции. След като времевият дял на един процес изтече, операционната система превключва на друг процес.

3. **Многозадачна операционна система (Multitasking Operating System): Многозадачната операционна система** е тип ОС, която **превключва между различни процеси**, за да създаде илюзията, че всички задачи се изпълняват едновременно. Тя използва методи за **планиране на процесите (process scheduling)**, за да реши кой процес трябва да бъде изпълняван в даден момент.

    - **Превключване на процеси (Context Switching)**: Основният механизъм на многозадачността е **превключването на контексти**. Това е процес, при който операционната система спира изпълнението на един процес, съхранява състоянието му и след това превключва на друг процес. По-късно, когато процесът получи отново достъп до CPU, системата възстановява неговото състояние и той продължава изпълнението си от мястото, където е спрял.
    - **Предимства на многозадачността**:
        - **Ефективно използване на ресурси**: Операционната система използва процесорното време ефективно, като не позволява процесорът да остане бездействащ, докато чака един процес да завърши.

- **Подобрено потребителско изживяване**: Многозадачността позволява на потребителите да изпълняват няколко приложения едновременно (например работа с текстов редактор, уеб браузър и музикален плеър едновременно).

- **Пример**: Когато потребител работи на компютъра си, той може да отваря няколко приложения (уеб браузър, текстов редактор, музикален плеър и т.н.), като всяко от тях представлява отделен процес. Операционната система разпределя време на всеки от процесите, така че те изглеждат като да се изпълняват едновременно.

4. **Видове многозадачност:**
   a) **Cooperative Multitasking (Кооперативна многозадачност)**:

   - При кооперативната многозадачност операционната система разчита на това, че всеки процес ще освободи процесорното време, когато приключи своята задача. Това означава, че всеки процес трябва "доброволно" да предаде контрола на следващия процес.

   - **Ограничение**: Ако един процес реши да не освобождава процесора, цялата система може да забави своята работа или дори да замръзне.

   - **Пример**: По-старите версии на операционни системи като Windows 3.1 и ранните версии на Mac OS използват кооперативна многозадачност.

   b) **Preemptive Multitasking (Принудителна многозадачност)**:

   - При принудителната многозадачност операционната система контролира разпределението на процесорното време и насилствено прекъсва процесите, когато техният времеви дял изтече. Това гарантира, че всеки процес получава равен достъп до процесора.

   - **Предимства**: Това прави системата по-стабилна и предотвратява ситуации, при които един процес монополизира ресурса.
   - **Пример**: Съвременни операционни системи като Windows, Linux и macOS използват принудителна многозадачност.

Кооперативната многозадачност (Cooperative Multitasking) и принудителната многозадачност (Preemptive Multitasking)

1. **Кооперативна многозадачност (Cooperative Multitasking):** При кооперативната многозадачност всеки процес сам **решава колко дълго ще използва CPU-то**, преди да освободи контрол и да даде възможност на друг процес да се изпълни. Процесите трябва да работят "кооперативно" и да се редуват доброволно.

   - **Как работи?**: Процесът, който се изпълнява, продължава, докато сам не реши да освободи CPU-то, като сигнализира на операционната система, че е готов да предаде контрола на друг процес.
   - **Ограничение**: Ако един процес реши да не освобождава процесора или изпълнява дълготрайни задачи, това може да доведе до проблеми с производителността и дори до замръзване на системата. Това прави кооперативната многозадачност по-малко надеждна за сложни многозадачни среди.
   - **Пример**: По-стари версии на операционни системи като **Windows 3.1** и **Mac OS Classic** използват кооперативна многозадачност. При тези системи, ако едно приложение спре да отговаря, това може да доведе до замръзване на цялата система.

2. **Принудителна многозадачност (Preemptive Multitasking):** При принудителната многозадачност **операционната система контролира разпределението на процесорното време** между процесите. Процесите не решават сами колко време ще използват процесора. Вместо това ОС принудително спира процеса, когато неговият "времеви дял" (time slice) изтече, и прехвърля управлението на друг процес.

   - **Как работи?**: Операционната система използва планиращ алгоритъм, който определя колко време ще получи всеки процес за изпълнение. Когато това време изтече, ОС спира процеса и дава контрола на друг процес. Това позволява по-добро управление на ресурсите и предотвратява проблемите, свързани с блокиране на системата.
   - **Предимства**: Принудителната многозадачност е по-надеждна и сигурна, тъй като операционната система има пълен контрол над процесите и може да гарантира, че всички процеси получават равен достъп до CPU-то. Това прави системата по-устойчива на грешки, тъй като никой процес не може да задържи процесора за твърде дълго време.
   - **Пример**: Съвременни операционни системи като **Windows**, **Linux** и **macOS** използват принудителна многозадачност. Това позволява по-добра производителност и стабилност, тъй като процесите не могат да блокират системата, когато техният времеви дял изтече.

3. **Основни разлики:**

   a) **Контрол над процесорното време**:
      - **Кооперативна многозадачност**: Процесите решават кога да освободят CPU-то.
      - **Принудителна многозадачност**: Операционната система контролира разпределението на процесорното време и може да принуди процесите да освободят CPU-то.

b) **Надеждност**:

- **Кооперативна многозадачност**: Може да доведе до замръзване или блокиране на системата, ако процес не освободи CPU-то.
- **Принудителна многозадачност**: По-надеждна, тъй като ОС може да принуди процесите да освободят ресурси.

c) **Приложимост**:

- **Кооперативна многозадачност**: Използвана в по-стари операционни системи.
- **Принудителна многозадачност**: Използвана в съвременни операционни системи за по-добро управление на ресурсите и стабилност.

Пръстени на защита (Protection Rings)

1. **Какво са пръстени на защита? - Пръстените на защита** са слоеста архитектура, която осигурява различни нива на **достъп до ресурси и привилегии в компютърната система**. Тази архитектура се използва за защита на данни и функционалности от грешки или злонамерени действия, като предоставя на различни части от системата достъп до различни нива на привилегии.

   - **Цел на пръстените на защита**: Да се изолира **основната функционалност на операционната система (ядрото)** от потребителските приложения и драйверите, така че проблеми в потребителските процеси да не засягат стабилността на системата като цяло.
   - **Хардуерна защита**: Тази архитектура се прилага и контролира от **процесора (CPU)**, който определя кой процес има достъп до различни хардуерни ресурси и привилегии, използвайки **микрокод (CPU microcode)**.

2. **Пръстени на защита:** Пръстените на защита обикновено са номерирани от **Ring 0** (най-високо ниво на привилегии) до **Ring 3** (най-ниско ниво на привилегии). Те създават слоеве на сигурност, като по-ниските пръстени имат повече права за директен достъп до хардуера, докато по-високите пръстени са ограничени в действията си.

Ring 0 (Пръстен 0) – Ядрото на операционната система (OS Kernel)

- **Ядрото на ОС** има най-високото ниво на привилегии и контролира всички ресурси на системата, включително достъп до хардуера. Този слой съдържа основни функции като управление на паметта, управление на процесите и взаимодействие с хардуера.
- **Пример**: **OS Kernel** (ядрото на операционната система), което отговаря за изпълнението на основните системни задачи. Ако нещо се обърка на това ниво, цялата система може да спре да работи.

Ring 1 и Ring 2 – Драйвери и системни услуги

- **Ring 1 и 2** обикновено се използват за драйвери или ниско ниво на системни услуги. Те имат достъп до определени хардуерни функции, но не толкова, колкото Ring 0.

- В много съвременни операционни системи Ring 1 и 2 не се използват активно, като драйверите и услугите се изпълняват или в пръстен 0, или в пръстен 3.

Ring 3 (Пръстен 3) – Потребителски приложения

- **Ring 3** има най-ниските привилегии и обикновено съдържа **потребителски приложения**, които се изпълняват в изолирана среда. Това означава, че приложенията нямат директен достъп до хардуера и трябва да се обръщат към операционната система за всякакъв вид взаимодействие с хардуерните ресурси.
- **Пример**: Приложения като **Nginx** и **Redis** се изпълняват в пръстен 3, където техният достъп до системните ресурси е ограничен и трябва да комуникират с ядрото за достъп до хардуера.

3. Защо са важни пръстените на защита?

a) **Изолация и сигурност**: Пръстените на защита осигуряват **изолация** между различните компоненти на операционната система и потребителските приложения. Това означава, че грешка или атака в едно приложение (например вирус или неправилно функциониращ драйвер) не може директно да засегне ядрото на системата или други критични части на системата.

b) **Предотвратяване на грешки**: Потребителските процеси в Ring 3 не могат да променят важни системни функции или директно да взаимодействат с хардуера. По този начин грешките, причинени от лошо написан код или зловреден софтуер, са ограничени до потребителското ниво и не могат да повлияят на ядрото или на хардуера.

c) **Контрол върху достъпа**: Хардуерът и системните ресурси се контролират стриктно чрез пръстените на защита. Програмите с по-ниски привилегии (в по-високите пръстени) трябва да поискат достъп до ресурси чрез ядрото, което добавя допълнителен слой на контрол и сигурност.

Режимите на работа на процесора: Kernel Mode (режим на ядрото) и User Mode (потребителски режим).

Kernel Mode (Режим на ядрото) и User Mode (Потребителски режим): Модерните операционни системи използват два основни режима на достъп до процесорните ресурси, за да защитят критични данни и да разделят изпълнението на системните функции от потребителските приложения. Тези два режима са **режим на ядрото** и **потребителски режим**.

1. **User Mode (Потребителски режим):** Потребителският режим (user mode) е режимът, в който потребителските приложения се изпълняват, обикновено в Ring 3. В този режим процесите имат ограничен достъп до системните ресурси и нямат право да взаимодействат директно с хардуера.
   - Какво е характерно за потребителския режим?:

- Процесите нямат достъп до критични ресурси като паметта на операционната система или хардуерните компоненти.
- Приложенията, които се изпълняват в потребителски режим, трябва да правят заявки към операционната система чрез системни повиквания (syscalls), когато имат нужда от достъп до ресурси.

- Предимства:
  - Ако възникне грешка в дадено приложение, изпълнявано в потребителски режим, тази грешка ще бъде изолирана и няма да повлияе на останалите системни компоненти или други процеси.
- Пример: Потребителски приложения като уеб браузъри, текстови редактори и игри се изпълняват в потребителски режим, където достъпът им до хардуера е строго контролиран и ограничен.

2. **Kernel Mode (Режим на ядрото):** Режимът на ядрото (kernel mode) е режимът с най-високо ниво на привилегии, обикновено свързан с Ring 0. В този режим операционната система и критични системни компоненти, като драйвери и системни услуги, имат пълен достъп до всички ресурси на компютъра, включително паметта и хардуера.
   - Какво е характерно за режима на ядрото?:
     - Операционната система и системните услуги имат пълен достъп до всички хардуерни ресурси на компютъра.
     - В режим на ядрото се изпълняват задачи като управление на паметта, управление на процесите, контрол на хардуерните устройства и драйверите.
     - Драйверите на устройства, които взаимодействат директно с хардуера, също се изпълняват в режим на ядрото.
   - Предимства:
     - Позволява директен достъп до всички ресурси и хардуер, което е необходимо за изпълнението на системните задачи.
   - Пример: Операционното ядро (OS Kernel), което контролира основните функции на операционната система, се изпълнява в режим на ядрото. Също така драйвери за устройства, като графични карти или мрежови адаптери, работят в този режим, за да имат достъп до хардуерните ресурси.

3. **Важност на режимите**

I. **Изолация и сигурност**: Един от основните принципи на тази архитектура е **сигурността**. Чрез разделяне на достъпа до системните ресурси, операционната система предотвратява потребителските приложения да извършват нежелани или опасни действия. Приложенията в потребителски режим не могат да нанесат сериозни щети на системата, тъй като имат ограничен достъп.

II. **Контрол на достъпа**: Операционната система действа като **посредник между хардуера и потребителските приложения**. Когато приложение иска достъп до хардуер, то трябва

да направи системно повикване и да се довери на операционната система да управлява този достъп по безопасен начин.

Модули или драйвер

1. **Устройство (Device): Устройството** е **хардуерен компонент**, който изпълнява специфична функция и е свързан към компютърната система. Устройствата могат да бъдат вътрешни (например графична карта, твърд диск) или външни (например мишка, клавиатура, принтер). Всяко устройство има своя специфична роля и начин на работа, който трябва да бъде разбран и контролиран от операционната система.
   - **Пример за устройства**:
     - **Твърд диск**: Служи за съхранение на данни.
     - **Графична карта**: Обработва визуалната информация и я предава на монитора.
     - **Мрежова карта**: Осигурява връзка на компютъра с интернет или локалната мрежа.

2. **Модул на ядрото или драйвер на устройство (Kernel Module or Device Driver): Драйверът на устройство (device driver)** е **софтуерна програма**, която управлява и контролира работата на определен тип устройство, свързано към компютърната система. Той действа като **посредник между операционната система и хардуера**, превеждайки заявките за вход/изход (I/O) от потребителските приложения към специфични инструкции, които устройството може да разбере и изпълни.
   - **Модул на ядрото (Kernel Module)**: В много операционни системи, драйверите се изпълняват като **модули на ядрото**. Това означава, че те са част от операционната система и работят в **режим на ядрото (kernel mode)**, което им позволява директен достъп до хардуера.
   - **Как работи драйверът?**:
     - Когато потребителското приложение направи заявка за взаимодействие с устройство (например запис на данни на твърдия диск), **драйверът** приема тази заявка и я превръща в конкретни команди за хардуера.
     - След това драйверът комуникира с устройството, за да осигури изпълнението на задачата.
     - Когато устройството завърши изпълнението на заявката, драйверът връща резултата на операционната система, която от своя страна го предава на приложението.

3. **Основни функции на драйверите и модулите на ядрото:**
   I. **Управление на I/O функции (Managing I/O Function Calls):** Драйверите управляват всички **входно/изходни операции** между приложението и устройството. Те осигуряват правилно взаимодействие между софтуера и хардуера, като превеждат командите от приложението в такива, които устройството може да изпълни.

      - **Пример**: Когато потребител запази файл, приложението отправя заявка за запис към операционната система. Драйверът на твърдия диск поема тази заявка, превръща я в команда за запис и я изпраща на твърдия диск, който изпълнява операцията.

   II. **Превод на потребителски заявки (Translating User I/O Requests):** Една от основните функции на драйверите е **превеждането на заявките за вход/изход от потребителския софтуер към хардуера**. Приложенията и операционната система работят на по-високо ниво на абстракция, докато устройствата работят на по-ниско, хардуерно ниво. Драйверът действа като "преводач" между тези две нива.

      - **Пример**: Когато потребителят изпрати команда за печат към принтер, драйверът на принтера превръща тази команда в инструкции, които принтерът може да разбере и изпълни.

   III. **Контрол на достъпа до хардуера (Controlling Access to Hardware)**

Драйверите осигуряват **сигурен и контролиран достъп до хардуера**, като гарантират, че само авторизирани процеси могат да комуникират с устройството и че няма конфликт в използването на устройството от множество процеси.

   - **Пример**: Когато няколко приложения се опитат да използват звуковата карта едновременно, драйверът гарантира, че тези заявки се управляват по правилен начин, за да не се получат конфликти.

4. **Важността на драйверите и модулите на ядрото**

   a) **Осигуряване на съвместимост между софтуер и хардуер**: Драйверите гарантират, че операционната система и приложенията могат да комуникират правилно с различни устройства, като правят хардуера достъпен за софтуера.

   b) **Сигурност и стабилност**: Като част от ядрото на операционната система, драйверите играят важна роля за стабилността на системата. Лошо написан драйвер може да доведе до срив на системата, тъй като работи в режим на ядрото и има пълен достъп до хардуера.

   c) **Оптимизация и производителност**: Добре написаните драйвери оптимизират взаимодействието между хардуера и софтуера, като подобряват производителността на устройствата.

Как операционната система идентифицира подходящия драйвер?

Операционната система използва специални **идентификатори на устройства (Device Identification String)**, за да разпознае хардуерните компоненти, свързани към компютърната система. Тези идентификатори съдържат информация за **производителя на устройството (vendor)** и **модела на устройството**, което позволява на операционната система да намери и зареди подходящия драйвер за това устройство.

1. **Идентификационен низ на устройство (Device Identification String):** Всяко компютърно устройство, свързано към системата, има **набор от регистри**, които съдържат уникални идентификационни номера, наречени **идентификатори на устройство**. Те включват:
   - **ID на производителя (Vendor ID)**: Уникален код, който идентифицира производителя на устройството.
   - **ID на устройството (Device ID)**: Уникален код, който идентифицира конкретния модел на устройството.

Операционната система използва тези **идентификатори**, за да разпознае устройството и да провери дали има инсталиран подходящ драйвер за него.

2. **Как работи процесът? -** Когато устройство е свързано към компютъра (например при стартиране на системата или когато ново устройство е добавено), **операционната система проверява идентификационните регистри** на устройството. Тези регистри съдържат **идентификационния низ на устройството**, който се състои от **Vendor ID** и **Device ID**.
   - **Процесът включва следните стъпки**:
     1. **Откриване на устройството**: Операционната система разпознава, че е свързано ново устройство.
     2. **Четене на идентификационния низ**: Операционната система чете идентификационния низ на устройството, който съдържа информация за производителя и модела.
     3. **Търсене на драйвер**: Операционната система проверява своята база данни от драйвери, за да намери съвместим драйвер на база прочетените идентификационни номера. Ако намери съвместим драйвер, той се зарежда и устройството започва да функционира.
     4. **Инсталиране на нов драйвер (ако е необходимо)**: Ако операционната система не намери подходящ драйвер в своята база данни, тя може да поиска от потребителя да инсталира нов драйвер, като го изтегли от интернет или използва предоставен драйвер от производителя.

3. **Пример за идентификационен низ на устройство:**
   - **PCI VEN_10E8&DEV_4750**

- **VEN_10E8**: Този идентификатор представлява производителя на устройството (Vendor ID). В този случай, „10E8“ е уникален код за конкретния производител.
- **DEV_4750**: Това е идентификаторът на устройството (Device ID), който указва конкретния модел на устройството.

Този идентификационен низ позволява на операционната система да разпознае точното устройство и да зареди съответния драйвер.

4. Защо е важен този процес?

a) **Автоматизация и лесно управление на хардуера**: Благодарение на този механизъм, операционната система може автоматично да разпознава и управлява устройства без необходимостта от ръчно инсталиране на драйвери за всяко устройство. Това значително улеснява управлението на хардуерни компоненти.

b) **Съвместимост**: Операционната система трябва да зареди правилния драйвер, за да гарантира, че устройството ще функционира правилно. Неправилно избраният драйвер може да доведе до грешки или нефункционални устройства.

c) **Поддръжка на множество устройства**: С този механизъм операционната система може да поддържа голямо разнообразие от устройства, включително различни модели от един и същ производител.

Разликите между сървърни операционни системи (Server Operating Systems) и десктоп операционни системи (Desktop Operating Systems)

1. Сървърни операционни системи (Server Operating Systems): Сървърните операционни системи са предназначени да управляват и обслужват множество потребители и системи едновременно. Те са създадени с акцент върху **стабилност, сигурност и управление на ресурси** и често се използват за управление на големи мрежи, бази данни и уеб сървъри.

- **Основни характеристики на сървърните ОС:**

1. **Управление на множество потребители и системи**: Сървърните операционни системи са проектирани да управляват **голям брой потребители** и да предоставят ресурси на множество машини в мрежа. Те могат да обслужват заявки от различни клиенти и потребители едновременно.

2. **Фокус върху стабилност, сигурност и управление на ресурси**:

   - **Стабилност**: Сървърните ОС трябва да работят без прекъсване за дълги периоди от време. Те трябва да могат да се справят с натоварване, без да се сриват.

   - **Сигурност**: Тъй като сървърите често се използват за съхранение на чувствителна информация или предоставяне на важни услуги, сигурността е основен приоритет. Сървърните ОС включват **разширени функции за сигурност** и защита от неоторизиран достъп.

- **Управление на ресурси**: Тези операционни системи са оптимизирани за ефективно използване на системните ресурси като процесорно време, памет и мрежови ресурси. Те трябва да могат да управляват множество заявки и да разпределят ресурсите оптимално.

3. **Оптимизирани за фонови задачи и услуги**: Сървърните ОС са проектирани да изпълняват **фонови задачи** като управление на бази данни, уеб сървъри и други критични услуги. Те не са фокусирани върху интерактивно потребителско изживяване, а върху **непрекъснато обслужване** на заявки.

4. **Примери за сървърни операционни системи**:

   - **Linux дистрибуции** като **Ubuntu Server**, **RedHat Enterprise Linux**, **Debian** и други. Тези системи се използват широко в корпоративни среди поради тяхната стабилност и гъвкавост.

   - **Windows Server**: Версия на Windows, специално разработена за сървъри и управление на мрежи и услуги.

   - **Unix-базирани системи** като **BSD** и **AIX** се използват често в критични корпоративни среди поради тяхната надеждност и сигурност.

2. **Десктоп операционни системи (Desktop Operating Systems): Десктоп операционните системи** са проектирани за **персонални** или **еднопотребителски задачи**. Те са фокусирани върху **потребителски интерфейс, лекота на използване и мултимедия** и предоставят интерактивно потребителско изживяване за задачи като сърфиране в интернет, обработка на документи и възпроизвеждане на мултимедия.

- **Основни характеристики на десктоп ОС:**

1. **Предназначени за персонални или еднопотребителски задачи**: За разлика от сървърните ОС, които трябва да обслужват множество потребители едновременно, **десктоп операционните системи** са проектирани за **лична употреба** от един потребител. Те са оптимизирани за изпълнение на приложения като текстообработващи програми, браузъри и игри.

2. **Фокус върху потребителски интерфейс и лекота на използване**:

   - Десктоп операционните системи са създадени да бъдат лесни за използване и **интуитивни**. Те включват **графични потребителски интерфейси (GUI)**, които улесняват потребителите при работа с файлове, папки и приложения.

   - **Лесен достъп до мултимедия и развлечения**: Тези операционни системи често са оптимизирани за задачи като **гледане на видео**, **слушане на музика** и **игри**, предоставяйки богато потребителско изживяване.

3. **Оптимизирани за интерактивни задачи**: Докато сървърните ОС са проектирани да изпълняват фонови задачи, **десктоп операционните системи** са оптимизирани за **интерактивни задачи**, като работа с браузъри, редактиране на документи и други дейности, които изискват непосредствена обратна връзка от системата.

4. **Примери за десктоп операционни системи**:

- **Windows**: Най-популярната десктоп операционна система, широко използвана за персонални компютри.
- **MacOS**: Операционната система на Apple, използвана на компютрите Mac, известна с интуитивния си интерфейс и силната интеграция с хардуера.
- **Linux дистрибуции** като **Ubuntu Desktop**, **Fedora** и други, които се използват за персонални компютри и предлагат безплатни и гъвкави решения за потребителите.

3. **Основни разлики между сървърни и десктоп операционни системи**

a) **Цел**:

- **Сървърните ОС** са предназначени да обслужват множество потребители и системи едновременно и да предоставят критични услуги като бази данни, мрежови услуги и уеб сървъри.
- **Десктоп ОС** са проектирани за персонални задачи и осигуряват лесен достъп до мултимедия, приложения и интернет.

b) **Фокус**:

- **Сървърните ОС** се фокусират върху **стабилност, сигурност и ефективно управление на ресурси**.
- **Десктоп ОС** се фокусират върху **удобство, лекота на използване и мултимедия**.

c) **Оптимизация**:

- **Сървърните ОС** са оптимизирани за изпълнение на **фонови задачи и услуги**, като целта е максимална производителност и минимални прекъсвания.
- **Десктоп ОС** са оптимизирани за **интерактивни задачи**, които изискват незабавна обратна връзка към потребителя.

d) **Сигурност и стабилност**:

- **Сървърните ОС** трябва да осигуряват **висока степен на сигурност** и **минимални прекъсвания** поради важната роля, която играят в мрежите и системите.
- **Десктоп ОС** предлагат сигурност, но приоритетът е удобството на потребителя и предоставянето на приятна работа с компютъра.

Съвременните парадигми на сървърните операционни системи

1. **Automatic Provisioning (Автоматично снабдяване): Автоматичното снабдяване** представлява механизъм, при който сървърите и ресурсите в дадена инфраструктура се **автоматично разгръщат**, за да посрещнат нуждите на приложението или системата, без да

се изисква ръчна намеса от администратора. Това включва както предоставяне на хардуерни ресурси, така и виртуални ресурси, като виртуални машини и контейнери.

- **Как се осъществява? -** Технологии като **CloudFormation** на AWS или **Terraform** позволяват автоматизирано разгръщане на сървъри, мрежи, бази данни и други инфраструктурни компоненти на базата на конфигурационни файлове. Тези файлове описват какви ресурси трябва да бъдат разположени и как да бъдат конфигурирани.
- **Процесът включва**:
  1. **Оценка на нуждите**: Системата следи натоварването и определя кога е нужно да се добавят нови ресурси.
  2. **Разгръщане**: Новите ресурси автоматично се създават и конфигурират според зададените правила.
  3. **Интеграция**: След разгръщане ресурсите автоматично се добавят към съществуващата инфраструктура.
- **Пример:** При използване на **автоматично снабдяване** в облачна среда като AWS, ако уебсайтът ви изпитва пик на трафик, системата автоматично ще разположи нови сървъри, за да се справи с увеличението на натоварването. Когато трафикът спадне, системата ще премахне ненужните сървъри, за да спести ресурси.

- **Ползи:**
  - ⇒ **По-бързо мащабиране**: Можете да добавяте и премахвате ресурси много бързо.
  - ⇒ **Автоматизация**: Намалява ръчната намеса и грешките, които могат да възникнат при ръчни конфигурации.
  - ⇒ **Ефективност на разходите**: Можете да разпределяте ресурси динамично, според нуждите на системата, без да разходвате повече ресурси от необходимото.

2. Orchestration (Оркестрация): Оркестрацията автоматизира управлението на множество взаимосвързани компоненти в рамките на сложна инфраструктура. Това е особено важно в **контейнерни среди**, където множество контейнери трябва да бъдат синхронизирани, за да работят заедно безпроблемно.

- **Как работи? -** Оркестрацията включва **координация на различни етапи** от жизнения цикъл на сървъра или контейнера, като разгръщане, конфигурация, мащабиране и наблюдение. В **Kubernetes**, например, оркестрацията гарантира, че ако контейнер се срине или спре, той автоматично се рестартира, за да се поддържа непрекъсната услуга.

- **Пример:** В среда с много микросервизи, всяка услуга може да бъде разположена в отделен контейнер. **Kubernetes** автоматизира разполагането и управление на контейнерите, като гарантира, че всички части на системата работят координирано и без прекъсвания.

- **Ползи:**
  - ⇒ **Автоматизация**: Оркестрацията премахва нуждата от ръчно управление на отделни компоненти.

⇒ **Устойчивост**: Ако даден компонент се провали, оркестрацията автоматично го възстановява.

⇒ **Скалиране**: Системите могат да се мащабират вертикално или хоризонтално според нуждите.

3. Configuration Management (Управление на конфигурации): Управлението на конфигурации е процесът, който гарантира, че инфраструктурата и приложенията са **последователно конфигурирани** по време на техния жизнен цикъл. Чрез автоматизация се гарантира, че всички сървъри и устройства следват еднакви правила за конфигурация, което улеснява поддръжката и управлението.

- **Как работи? -** Системите за управление на конфигурации като **Puppet**, **Chef** или **Ansible** използват **скриптове** или **конфигурационни файлове**, за да задават какви пакети трябва да се инсталират, как трябва да бъде конфигуриран софтуерът и какви права да има всеки потребител.

- **Пример:** Ако имате множество сървъри, които изискват инсталация на специфичен софтуер и конфигурация на мрежови настройки, чрез Ansible можете да автоматизирате този процес за всички сървъри едновременно.

- **Ползи:**

  ⇒ **Уеднаквяване на средата**: Всички сървъри имат идентична конфигурация, което намалява риска от грешки.

  ⇒ **Проследимост**: Всички промени в конфигурацията се проследяват и могат лесно да бъдат възстановени при нужда.

  ⇒ **Сигурност**: Уверявате се, че всички машини следват най-добрите практики за сигурност.

4. Self-Healing (Автоматично възстановяване): Автоматичното възстановяване осигурява устойчивостта на системата чрез откриване и автоматично коригиране на проблеми. Това значително подобрява **времето за непрекъсната работа (uptime)** на системата, тъй като грешките се поправят автоматично, без нужда от намеса на оператор.

- **Как работи? - Механизми за наблюдение** следят за аномалии, като срив на приложения или прекъсване на услуга, и автоматично предприемат действия за възстановяване. Например, контейнерите могат да бъдат автоматично рестартирани, ако възникне грешка, или задачи могат да бъдат преразпределени към други възли.

- **Пример:** В облачна платформа като **AWS** с услугата **Auto Healing**, ако сървър спре да отговаря, автоматично се стартира нов сървър, който поема задачите на повредения сървър.

- **Ползи:**

  ⇒ **Намалява прекъсванията**: Проблемите се решават без човешка намеса, което минимизира времето за престой.

⇒ **Подобрена надеждност**: Системите могат да се самовъзстановяват и остават стабилни дори при неочаквани събития.

5. Immutability (Непроменливост): Непроменливостта е подход, при който сървърите и компонентите не се променят след първоначалното си разгръщане. Вместо да се актуализират съществуващите сървъри, те се заменят с нови версии.

- **Как работи? -** Когато се изисква промяна в конфигурацията или софтуера, вместо да се актуализира съществуващата среда, създава се **нова инстанция** с промените и старата се премахва. Това елиминира възможността от неправилно актуализиране или конфликти при инсталиране на нов софтуер.
- **Пример:** Системи като **Docker** следват тази практика чрез разгръщане на нови контейнери за всяка версия на приложението, което позволява бързо и предсказуемо разгръщане.
- **Ползи:**

  ⇒ **Стабилност**: Намалява риска от конфликти и грешки при актуализация.

  ⇒ **Предсказуемост**: Винаги се знае каква е състоянието на системата, тъй като всеки нов компонент е тестван и валидиран преди разгръщането.

6. Infrastructure as Code (IaC): Инфраструктурата като код (IaC) е практика, при която цялата инфраструктура се дефинира и управлява чрез **програмни скриптове** или **конфигурационни файлове**. Вместо инфраструктурата да се управлява ръчно, всеки аспект от нея се описва като код, което позволява автоматизация и повторяемост.

- **Как работи?** - Инструменти като **Terraform** или **CloudFormation** позволяват на администраторите да описват ресурси като виртуални машини, мрежи, бази данни и други като код. Този код може да бъде изпълняван, за да създаде или промени инфраструктурата автоматично.

- **Пример:** Ако искате да разположите нова облачна инфраструктура за уеб приложение, можете да опишете всички нужни ресурси (сървъри, мрежи, бази данни) като код и да ги разположите автоматично чрез изпълнение на този код.

- **Ползи:**

  ⇒ **Автоматизация и повторяемост**: Веднъж дефинирана инфраструктурата като код, тя може да бъде разположена многократно, без риск от грешки.

  ⇒ **Проследимост**: Всички промени в инфраструктурата могат да бъдат проследявани и възстановявани при нужда.

Източници:

- Презентации от магистърски курс “DevOps автоматизация в облачните технологии”, ФМИ, СУ, 2024;
- ChatGPT -4o