2. EA

Regiszter: A regiszterek a számítógépek központi feldolgozó egységeinek (CPU-inak), illetve mikroprocesszorainak gyorsan írható-olvasható, ideiglenes tartalmú, és általában egyszerre csak 1 gépi szó (word) (rövid karakterlánc, 1-2 szó általában 2-4 bájt) feldolgozására alkalmas tárolóegységei.

Processzor védelmi szintek:
Intel 80286 – minden utasítás egyenlő
Intel 80386 – 4 védelmi szint, ebből 2-őt használ, kernel mód (védett, protected mód) és felhasználói mód

Megszakítások:
Szoftveres megszakítás, csapda(trap) kezelése azonos a hardveres megszakításkezeléssel.
Megszakítások nagyon fontos elemei a számítógépek működésének. Amikor a mikroprocesszornak egy eszközt, vagy folyamatot ki kell szolgálni, annak eredeti tevékenységét felfüggesztve, megszakítások lépnek életbe. Létezik szoftveres, és hardveres megszakítás. Hardveres megszakítás esetén a processzor erőforrást (processzoridő) igénylő eszköz megszakításkérést a mikroprocesszor megszakítás engedélyezés kérés vezetékéhez. Amennyiben a megszakítás lehetséges, a kérést kezdeményező eszköz használhatja a mikroprocesszort. Szoftver megszakítás esetén a főprogram futását egy alprogram szakítja meg. Ebben az esetben a főprogram futásállapota elmentésre kerül, majd miután a megszakítást kérő program befejezte a műveletet a főprogram folytatja a futását a megszakítás előtti pozícióból.

Feladat maszkolása: megszakítások engedélyezése, letiltása.

Nem maszkolható feladatok (NMI): pl. Súlyos hardver hiba, például memóriahiba, vagy tápfeszültség kimaradás esetén keletkezik

Operációs rendszer: olyan program ami egyszerű felhasználói felületet nyújt, eltakarva a számítógép (rendszer) eszközeit.

Kommunikáció a perifériákkal:
- Lekérdezéses átvitel (polling) – folyamatos lekérdezés
- Megszakítás (Interrupt) – nem kérdezgetjük folyamatosan, hanem az esemény bekövetkezésekor a megadott programrész kerül végrehajtásra. Pl.: aszinkron hívások esetén (aszinkron hívások: Olyan adatátviteli mód, amikor a két kommunikáló fél nem használ külön időzítő jelet, ellentétben a szinkron átvitellel. éppen ezért szükséges az átvitt adatok közé olyan információ elhelyezése, amely megmondja a vevőnek, hogy hol kezdődnek az adatok)
- DMA: közvetlen memória elérés (6. EA-ban van részletezve)

API:
Az alkalmazásprogramozási felület vagy alkalmazásprogramozási interfész (angolul application programming interface, röviden API) egy program vagy rendszerprogram azon eljárásainak (szolgáltatásainak) és azok használatának dokumentációja, amelyet más programok felhasználhatnak. Egy nyilvános API segítségével lehetséges egy programrendszer szolgáltatásait használni anélkül, hogy annak belső működését ismerni kellene. Általában nem kötődik programozási nyelvhez. Az egyik leggyakoribb esete az alkalmazásprogramozási felületnek az operációs rendszerek programozási felülete: annak dokumentációja, hogy a rendszeren futó programok milyen – jól definiált, szabványosított – felületen tudják a rendszer szolgáltatásait használni.

POSIX:
Valójában egy minimális rendszerhívás (API) készlet szabvány, aminek témaköreibe tartozik pl.: fájl, könyvtárműveletek, folyamatok kezelése, szignálok, szemaforok stb. Ma gyakorlatilag minden OS POSIX kompatibilis.

Firmware: Hardverbe a gyártó által épített szoftver (merevlemez, billentyűzet, monitor, memóriakártya, de pl. távirányító vagy számológép is)

Middleware: Operációs rendszer feletti réteg

Operációs rendszer generációk:
- Történelmi generáció: mechanikus, nincs oprendszer, operátoralkalmazás
- Első generáció: kapcsolótábla, vákuumcső (Neumann János)
- Második generáció: tranzisztoros rendszer, lyukkártyás, szalagos gépek, oprendszer, fortan nyelv, - kötegelt rendszer megjelenése (5. EA-ban van részletezve)
- Harmadik generáció: integrált áramkörök, azonos rendszerek, kompatibilitás megjelenése, mutliprogramozás, multitask megjelenése, időosztás megjelenése.
- Negyedik generáció: napjainkban

Rendszerhívások: azok a szolgáltatások amelyek az operációs rendszer és a felhasználói programok közötti kapcsolatot biztosítják. Két fajtája: processz kezelő, fájlkezelő csoport.

Processz: egy végrehajtás alatt lévő program. Saját címtartománnyal rendelkezik, megszüntetés, felfüggesztés, és processzek kommunikációja is lehetséges. Processz táblázat: cím, regiszter, munkafájl adatok.

Operációs rendszer struktúrák:
- Monolitikus rendszerek: nincs különösebb struktúrája, rendszerkönyvtár egyetlen rendszer, így mindenki mindenkit láthat. Információelrejtés nem igazán van. Létezik modul, modulcsoportos tervezés. Rendszerhívás során gyakran felügyelt (kernel) módba kerüll a CPU. Paraméterek a regiszterekben, valamint a csapdázás (trap) is jellemző.
- Rétegelt szerkezet:
    5. Gépkezelő
    4. Felhasználói programok
    3. Bemeneti / Kimenet kezelése
    2. Gépkezelő-folyamat
    1. Memória és dobkezelés
    0. Processzorhozzárendelés és multiprogramozás

3. EA

Mágnesszalagok: sorrendi, lineáris felépítés, keretek rekordokba szerveződnek. Rekordok között rekord elválasztó (record gap) Nem igazán olcsó, 800/1600 GB férőhely. Biztonsági mentésekre, nagy mennyiségű adat tárolására használatos.

FDD (floppy)**, HDD** – kör alakú lemez – sávos felosztás, sávok szektorokra bonthatók - blokk (klaszter – több blokk, a fájlrendszer (fájlrendszerekről még ebben az EA-ban van szó) által megválasztott logikai tárolási egység). Több lemez – egymás alatti sávok: cilinder (HDD-nél)

Optikai tárolók: CD, DVD – fényvisszaverődés alapján

Eszközmeghajtó (Device driver): Az a program, amely a közvetlen kommunikációt végzi. A kernel, az operációs rendszer magjának része. Lemezek írása – olvasása során DMA-t használnak. (DMA: 6. EA-ban van kifejtve), réteges felépítés.

Mágneslemez formázása: sávos-szektoros rendszer kialakítása.
- Quick format – Normal format: normal format hibás szektorokat is keres.
- Alacsony szintű formázás: szektorok kialakítása (ez gyártóknál elérhető)
- Logikai formázás: a partíciók kialakítása – max 4 logikai rész alakítható ki (oprendszer)

0. Szektor – MRB (Master Boot Record): partíciós szektor a merevlemez legelső szektorának (azaz az első lemezfelület első sávjának első szektorának) elnevezése. Csak a particionált merevlemezeknek van MBR-jük. A MBR a merevlemez legelején, az első partíció előtt található meg. Gyakorlatilag a merevlemez partíciók elhelyezkedési adatait tárolja.

Boot folyamat: ROM-BIOS megvizsgálja, lehet-e operációs rendszert betölteni, ha igen betölti a lemez MBR programját a 7c00h címre. Ez után az MRB programja vizsgálja meg mi az elsődleges partíció, majd azt betölti a memóriába.

Sorrendi ütemezés (FCFS – First Come First Service): Ahogy jönnek a kérése, úgy sorban szolgáljuk ki azokat. Biztosan minden kérés kiszolgálásra kerül, de nem törődik a fej aktuális helyzetével, kicsi az adatátviteli sebesség, és ezért nem igazán hatékony. Átlagos kiszolgálási idő, kis szórással.

„Leghamarabb először" ütemezés (SSTF – Shortest Sekk Time First): a legkisebb fejmozgást részesíti előnyben, átlagos várakozási idő kicsi, átviteli sávszélesség nagy, fennáll a kiéheztetés veszélye.

Pásztázó ütemezés (SCAN) módszer: a fej állandó mozgásban van, és a mozgás útjába eső kéréseket kielégíti. A fej mozgás megfordul ha a mozgás irányában nincs kérés, vagy a fej szélső pozíciót ér el. Rossz ütemben érkező kérések kiszolgálása csak oda – vissza mozgás után kerül kiszolgálásra. Középső sávok elérés szórása kicsi.

Egyirányú pásztázás (C-SCAN): csak egyirányú mozgás, ezért gyorsabb a fejmozgás, nagyobb sávszélesség, átlagos

várakozási idő hasonló, mint a SCAN esetén, viszont a szórás kicsi. Nem igazán fordulhat elő rossz ütemű kérés.
Ütemezés javítások: FCFS esetében ha az aktuális sorrendi kérés kiszolgálás helyén van egy másik kérés is akkor szolgáljuk ki azt is (Pick up)

Lemezek megbízhatósága: adatok redundáns tárolása úgy, hogy lemezsérülés esetén is adatvesztés-mentes maradjon.

Ütemezés javítása memória használattal:
- DMA maga is memória (6. EA)
- Memória puffer – Olvasás: ütemező feltölti, felhasználói program kiüríti. Írás: felhasználói folyamat tölti, ütemező kiüríti.
- Disc cache – lemez gyorsítótár

Dinamikus kötet: logikai meghajtó több lemezre helyezése

RAID (ha oprendszer nyújta SoftRaid-nek is nevezzük, ha külső vezérlőegység akkor Hardver Raid)
RAID 0 – több lemez logikai összefűzésével egy meghajtót kapunk, ezek összege adja az új meghajtó kapacitását. A logikai meghajtó blokkjait szétrakja a lemezekre, ezáltal egy fájl írása több lemezre kerül. Gyorsabb I%O műveletek, de nincs meghibásodás elleni védelem.
RAID 1 – Két független lemezből készít egy logikai egységet, minden adatot párhuzamosan kiír mindkét lemezre (tükrözés, mirror). Tárolókapacitás a felére csökken, drága megoldás, csak mindkettő lemez egyszerre történő meghibásodása esetén okoz adatvesztést.
RAID 2 – Adatbitek mellett hibajavító biteket is tartalmaz. Azaz +1 hibajavító diszk.
RAID 3 – Plusz paritásdiszk. Azaz +1 diszk
RAID 4 – A RAID 0 kiegészítése paritásdiszkkel
RAID 5 – Nincs paritásdiszk, ez el van osztva az összes elemére (stripe set). Adatok is elosztva tárolódnak. Intenzív CPU igény, két lemez egyidejű meghibásodása esetén okoz adatvesztést. Azaz +1 diszket igényel.
RAID 6 – A RAID 5 kiegészítése paritásdiszkkel. Azaz tárolóhely +2 diszk szükséges.

Megjegyzés: leggyakrabban az 1, 5 verziókat használják, a 6-os vezérlők az utóbbi 1-2 évben jelentek meg.

Fájl: adatok egy logikai csoportja, névvel, paraméterrel ellátva.

Könyvtár: fájlok logikai csoportosítása.

Fájlrendszer: módszer a fizikai lemezünkön, kötetünkön a fájlok és könyvtárak elhelyezés rendszerének kialakítására.

Elhelyezkedési stratégiák:
- Folytonos tárkiosztás
- Láncolt elhelyezkedés
- Indextáblás elhelyezkedés: katalógus tartalmazza a fájlhoz tartozó kis tábla (inode) címét. Ebből elérhető a fájl.

Naplózott fájlrendszer: sérülés, áramszünet, stb. esetén helyreállítható, nagyobb erőforrás idényű de jobb a megbízhatósága.

Fájlrendszerek:
- FAT: a FAT tábla a lemez foglalási térképe, annyi eleme van ahány blokk a lemezen. A katalógusban a fájl adatok (név stb) mellett csak az első fájl blokk sorszáma van megadva. A FAT blokk azonosító mutatja a következő blokk címét, ha nincs ilyen akkor FFF az érték. (vagyis láncolt elhelyezésben van). A fájl utolsó módosítási ideje is tárolva van. Töredezettségmentesítés szükséges.
- NTFS: kifinomult biztonsági beállítások, POSIX támogatás, tömörített fájl, mappa, felhasználói kvóta kezelés, ezen felül az NTFS csak klasztereket tart nyilván, szektort nem. Az NTFS partíció az MTF táblázattal kezdődik. Ha a fájl < 1kb akkor közvetlen elérést biztosít. Töredezettségmentesítés szükséges.
- UNIX könyvtárszerkezet: Indextáblás megoldás, boot blokk után a partíció szuperblokkja következik, ezt követi a szabad terület leíró rész (i-node tábla, majd gyökérkönyvtár bejegyzéssel). Moduláris elhelyezés, gyorsan elérhető az információ, sok kicsi táblázat, ez alkotja a katalógust. Egy fájlt egy i-node ír le.

4. EA

Valódi-e a Multi Task? - Nem. Csupán processzek közötti kapcsolgatás. Egy időben csak egy folyamat aktív.

1 feladat-végrehajtás=1 processzor+1 rendszer memória+1 I/O eszköz

Környezetváltásos rendszer: csak az előtérben lévő alkalmazás fut

Kooperatív rendszer: az aktuális processz bizonyos időközönként, vagy időkritikus műveletnél önként lemond a CPU-ról (Win3.1)

Preemptív rendszer: az aktuális processz től a kernel bizonyos idő után elveszi a vezérlést, és a következő várakozó folyamatnak adja. (ma tipikusan ilyen rendszereket használunk)

Real time rendszer: igazából ez is preemtív rendszer (különbségek később)

Folyamatok létrehozásának oka: boot, folyamatot eredményező rendszerhívás(fork, execve), felhasználói kérés (parancs&), nagy rendszerek kötegelt feladata

Folyamatok kapcsolata: szülő-gyerek kapcsolat, folyamatfa: 1 folyamat-1 szülő, 1 foly.-több gyerek,összetartozó folyamatcsoport.

Reinkarnációs szerver: meghajtó programok, kiszolgálók elindítója. Ha elhal az egyik, akkor azt újraszüli, reinkarnálja.

Folyamatok befejezése: megadott időkeret után. ok: önkéntes és önkéntelen

Önkéntes befejezés: szabályos kilépés (exit, return), program által felfedezett hiba miatt

Önkéntelen befejezés: illegális utasítás, végzetes hiba (|0). Külső segítséggel, netán mi lőjük ki.

Folyamat: önálló programegység, utasításszámlálóval, veremmel stb. Általában nem független folyamatok. Három állapotban lehet: futó, futásra kész, vagy blokkolt. (állapotátmenet diagram)

Folyamat megvalósítása: 1 aktív folyamat 1x-re. Mindent meg kell őrizni(regiszter, utasítás számláló, nyitott file infó).Globálisan maszkolható, a nem maszkolható le van kötve. Attól lesz egyszerű, hogy ezen folyamatokból válogatva valósul meg, valamint a taszk szegmens regiszterekkel mutat a táblákra, így a proci közvetlen támogatást ad a processzéhez.

Folyamatok váltása: időzítő, megszakítás, esemény, rendszerhívás kezdeményezés. CASH nem menthető.

Szál: egy folyamaton belül több egymástól „független” végrehajtási sor. A folyamatnak önálló címtartománya van, szálnak viszont nincs.

Folyamat-szál: csak a folyamatnak (címtartomány, glob.vált., megnyitott file leírók, gyermek foly., szignálkezelők…). Szálnak is: utasításszámláló, regiszterek, verem.

Folyamatleíró táblázat (PCB): A rendszer inicializálásakor jön létre. 1 elem, a rendszerleíró már bent van, amikor a rendszer elindul. Tömbszerű szerkezet (PID alapon) – de egy – egy elem egy összetett processzus adatokat tartalmazó struktúra. Egy folyamat fontosabb adatai: azonosítója, neve, tulajdonos, csoport stb.

Szálprobléma: Fork: gyerekben kell több szál, ha a szülőben több van. Filekezelés: egy szál lezár egy másik használta file-t. Hibakezelés: globális hibajelző, folyamatonként 1. A rendszerhívásoknak kezelni kell a szálakat!

IPC: (Inter Process Comunication) **Klaszterek:** megbízhatóság növelése a fő cél.

Versenyhelyzet: két vagy több folyamat közös memóriát ír vagy olvas.

Kritikus programterület: az a rész, amikor a közös erőforrást (memóriát) használjuk.

Kölcsönös kizárás: (szemléletes ábra 4. EA 21.dia)
A jó kölcsönös kizárás az alábbi feltételeknek felel meg:
- Nincs két folyamat egyszerre a kritikus szekciójában.
- Nincs sebesség, CPU paraméterfüggőség.
- Egyetlen kritikus szekción kívül levő folyamat sem blokkolhat másik folyamatot.
- Egy folyamat sem vár örökké, hogy a kritikus szekcióba tudjon belépni.

Megvalósítás:
- Megszakítások tiltása, zárolós változó használata, szigorú váltogatás

Peterson: javítása: kritikus szekció előtt belépés, illetve kilépés függvény meghívása

TSL: (Test and Set Lock), megszakíthatatlan atomi művelet. Peterson 1x-bb változata

Alvás-ébredés: blokkoljuk a folyamatot, amíg meg nem engedett (sleep–wakeup,down-up,stb…),Pl. Gyártó-fogyasztó.

Szemafor: egyfajta „kritikus szakasz védelem”. A szemafor tilosat mutat, ha értéke 0. Ha értéke >0 akkor az adott folyamat beléphet a kritikus területre. (Mi a „baj” a szemaforokkal? - könnyen el lehet rontani a kódolás során). Down és up művelet tartozik hozzá. (Dijkstra P és V műveletek)

Elemi művelet: a szemafor változó ellenőrzése, módosítása, megszakíthatatlan, versenyhelyzetet megelőzi (0 vagy 1)

5. EA

Monitor: Hasonló a szemaforhoz, de itt eljárások, adatszerkezetek lehetnek. Egy időben cska egy folyamat lehet aktív a monitoron belül. Megvalósítása mutex (lásd: köv. fogalom) segítségével történik. Apró gond: mi van ha egy folyamat nem tud továbbmenni a monitoron belül? Erre jók az állapot változók (condition). Rajtuk két művelet végezhető – wait vagy signal.
A monitoros megoldás egy vagy több VPU esetén is jó, de csak egy közös memória használatánál. Ha már önálló saját memóriájuk van a CPU-knak (dedikált memória) akkor ez a megoldás nem az igazi.

Mutex: A mutex egyik jelentése az angol mutual exclusion (kölcsönös kizárás) szóból ered. Programozástechnológiában Párhuzamos folyamatok használatakor előfordulhat, hogy két folyamat ugyanazt az Erőforrást (resource) egyszerre akarja használni. Ekkor jellemzően felléphet Versengés. Ennek kiküszöbölésére a gyorsabb folyamat egy, az erőforráshoz tartozó mutexet zárol (ún. lock-ol).
Amíg a mutex zárolva van (ezt csak a zároló folyamat tudja feloldani - kivéve speciális eseteket), addig más folyamat nem férhet hozzá a zárolt erőforráshoz. Így az biztonságosan használható. (Például nem lenne szerencsés, ha DVD-írónkat egyszerre két folyamat használná.

A szemafor és a mutex közti különbség: Az a különbség, hogy míg utóbbi csak kölcsönös kizárást tesz lehetővé, azaz egyszerre mindig pontosan csakis egyetlen feladat számára biztosít hozzáférést az osztott erőforráshoz, addig a szemafort olyan esetekben használják, ahol egynél több - de korlátos számú - feladat számára engedélyezett a párhuzamos hozzáférés.

Üzenetküldés: A folyamatok jellemzően két primitívet használnak: send (célfolyamat, üzenet) és receive(forrás, üzenet) – a forrás tetszőleges is lehet.

Nyugtázó üzenet: Ha a küldő és a fogadó nem azonos gépen van akkor szükséges egy úgynevezett nyugtázó üzenet. Ha ezt a küldő nem kapja meg, akkor ismét elküldi az üzenetet, ha a nyugta veszik el a küldő újra küld. Ismételt üzenetek megkülönböztetésére sorszámot használ.

Randevú stratégia: Üzenetküldésnél ideiglenes tárolók is jönnek létre mindkét helyen (levelesláda). Ezt el lehet hagyni, ekkor a send előtt van receive, a küldő blokkolódik illetve fordítva. - ez a randevú stratégia.

Ütemező: eldönti melyik folyamat fusson, az ütemezési algoritmus alapján.

Folyamat tevékenységei:
Vagy számolgat magában, vagy I/O igény, írni olvasni akar adott perifériára. Ez alapján megkülönböztetünk:
- Számításigényes feladat: hosszan dolgozik, keveset vár I/O – ra
- I/O igényes feladat: rövideket dolgozik, hosszan vár I/O-ra

Ütemezések csoportosítása: Minden rendszerre jellemző: pártatlanság, mindenki hozzáfér a CPU-hoz, mindenkire ugyanazok az elvek érvényesek, mindenki azonos terhelést kapjon.
Fajtái: kötegelt rendszerek, interaktív rendszerek, valós idejű rendszerek.

Kötegelt rendszerek (áteresztőképesség, áthaladási idő, CPU kihasználtság):
- Sorrendi ütemezés (FCFS): - nem megszakítható
  Egy folyamat addig fut, amíg nem végez vagy nem blokkolódik. Egy pártatlan, egyszerű láncolt listában tartjuk a folyamatokat, ha egy folyamat blokkolódik, akkor a sor végére kerül.
- Legrövidebb feladat először (SJB): - nem megszakítható
  Kell előre ismerni a futási időket, akkor optimális ha a kezdetben mindenki elérhető.
- Legrövidebb maradék futási idejű következzen: - megszakítható
  Minden új belépéskor vizsgálat.
- Háromszintű ütemezés:
  + Bebocsátó ütemező: a feladatokat válogatva engedi be a memóriába.
  + Lemez ütemező: ha a bebocsátó sok folyamatot enged be és elfogy a memória, akkor lemezre kell írni valamennyit, meg vissza. - ez ritkán fut.
  + CPU ütemező: a korábban említett algoritmusok közül választhatunk.

Interaktív rendszerek (válaszidő, megfelelés a felhasználói igényeknek)
- Körben járó ütemezés (Round Robin): Mindenkinek időszelet, aminek a végén, vagy blokkolás esetén jön a következő folyamat. Időszelet végén a lista végére kerül az aktuéális folyamat, ami pártatlan és egyszerű. Gyakorlatilag egy listában tároljuk a folyamatokat és ezen megyünk körbe – körbe. A legnagyobb kérdés hogy mekkora legyen egy időszelet? Mivel a processz átkapcoslás időigényes, ezért ha kicsi az időszelet sok CPU megy el a kapcsolgatásra, ha túl nagy akkor esetleg az interaktív felhasználóknak lassúnak tűnhet pl. a billentyűkezelés.
- Prioritásos ütemezés: Fontosság, prioritás bevezetése, a legmagasabb prioritású futhat. Prioritási osztályokat használ, ezeken belül az előbb említett Round Robin fut. Minden 100 időszeletnél újraértékeli a prioritásokat, különben nagy lenne a kiéheztetés veszélye.
- Többszörös sorok: Szintén prioritásos és Round Robinnal működik, a legmagasabb szinten minden folyamat 1 időszeletet kap, majd 2,4,16,32,64-et, ha elhasználta a legmagasabb szintű folyamat az idejét egy szinttel lejjebb kerül.
- Legrövidebb folyamat előbb: becslés alapján az előzőekből.
- Garantált ütemezés: minden aktív folyamat arányos CPU időt kap, nyilván kell tartani, hogy egy folyamat már mennyi időt kapott, ha valaki arányosan kevesebbet akkor az kerül előre.
- Sorsjáték ütemezés: Mint az előző, csak a folyamatok között „sorsjegyeket” osztunk szét, az kapja a vezérlést akinél a kihúzott jegy van.
- Arányos ütemezés: Mint a garantált, csak felhasználókra vonatkoztatva.

Valós idejű rendszerek: (határidők betartása, adatvesztés,minőségromlás elkerülése)
Az idő a kulcsszereplő, garantálni kell adott határidőre a tevékenység, válasz megoldását.
- Hard Real Time (szigorú) abszolút nem módosítható határidők.
- Soft Real Time (toleráns) léteznek határidők, de ezek kis méretű elmulasztása tolerálható.
A programokat kisebb folyamatokra bontják.

Szálütemezés:
- Felhasználói szintű szálak: kernel nem tud róluk, a folyamat kap időszeletet, ezen belül a szálütemező dönt ki fusson, gyors váltás a szálak között, és alkalmazásfüggő szálütemezés lehetséges.
- Kernel szintű szálak: Kernel ismeri a szálakat, kernel dönt melyik folyamat szála következzen. Lassú váltás, két szál váltása között teljes környezetátkapcsolás kell.

6. EA

I/O eszközök:
- Blokkos eszközök: adott méretű blokkokban tároljuk az információt, egymástól függetlenül írhatók vagy olvashatók, illetve blokkonként címezhető. Ilyen pl. HDD, és szalagos egység.
- Karakteres eszközök: nem címezhető, csak jönnek – mennek sorban a bájtok.
- Időzítő: kivétel, nem blokkos és nem is karakteres.

Megszakítások: Általában az eszközöknek van állapotbitjük, jelezve, hogy az adat készen van. Ezt lehet figyelni, nem az igazi. Tevékeny várakozás ez is, nem hatékony.
- Megszakítás használat (IRQ): CPU tevékenység megszakítása, nem kell állandóan lekérdezgetni.

Közvetlen memória elérés (DMA):
Tartalmaz: memória cím regisztert, átviteli irány jelzésére, mennyiségre regisztert. Ezeket szabályos I/O portokon lehet elérni.
- Működésének lépései:
1. CPU beállítja a DMA vezérlőt (regisztereket)
2. A DMA a lemezvezérlőt kéri a megadott műveletre.
3. Miután a lemezvezérlő beolvasta a pufferébe, a rendszersínen keresztül a memóriába(ból) írja, olvassa az adatot.
4. Lemezvezérlő nyugtázza, hogy kész a kérés teljesítése.
5. DMA megszakítással jelzi, befejezte a műveletet.

I/O szoftverrendszer felépítése:
Réteges szerkezet (tipikusan 4 réteg)
Hardver eszköz:
1. megszakítást kezelő réteg – legalsó kernel szinten kezelt szemafor blokkolással védve
2. eszközmeghajtó programok
3. eszköz független operációs rendszer program
4. felhasználói I/O eszközt használó program

Eszközmeghajtó programok (driver): Pontosan ismeri az eszköz jellemzőit, feladata a felette lévő szintről érkező absztrakt kérések kiszolgálása. Kezeli az eszközt I/O portokon, megszakítás kezelésen keresztül.

Holtpont (deadlock): Két vagy több folyamat egy erőforrás megszerzése során olyan helyzetbe kerül, hogy egymást blokkolják a további végrehajtásban. Vagyis folyamatokból álló halmaz holtpontban van, ha minden folyamat olyan másik eseményre vár, amit csak a halmaz egy másik folyamata okozhat. (Nem csak I/O eszközöknél, hanem jellemző pl. párhuzamos rendszereknél, adatbázisoknál stb.)

Holtpont feltételek:
1. Kölcsönös kizárás feltétel – minden erőforrás hozzá van rendelve 1 folyamathoz vagy szabad.
2. Birtoklás és várakozás feltétel. - Korábban kapott erőforrást birtokló folyamat kérhet újabbat.
3. Megszakíthatatlanság feltétel. - Nem lehet egy folyamattól elvenni az erőforrást, csak a folyamat engedheti el.
4. Ciklikus várakozás feltétel. - Két vagy több folyamatlánc kialakulása, amiben minden folyamat olyan erőforrásra vár, amit egy másik tart fogva.

Holtpont stratégiák:
1. A probléma figyelmen kívül hagyása. -Nem törődünk vele, nagy valószínűséggel Ő sem talál meg bennünket, ha mégis …
   Ezt a módszert gyakran strucc algoritmus néven is ismerjük. Kérdés, mit is jelent ez, és milyen gyakori probléma? Vizsgálatok szerint a holtpont probléma és az egyéb (fordító,oprendszer, hw, swhiba) összeomlások aránya 1:250. A Unix, Windows világ is ezt a „módszert használja.
2. Felismerés és helyreállítás. - Engedjük a holtpontot megjelenni (kör), ezt észrevesszük és cselekszünk.
   Folyamatosan figyeljük az erőforrás igényeket, elengedéseket. Kezeljük az erőforrás gráfot folyamatosan. Ha kör keletkezik, akkor egy körbeli folyamatot megszüntetünk. Másik módszer, nem foglalkozunk az erőforrás gráffal, ha x (fél óra?) ideje blokkolt egy folyamat, egyszerűen megszüntetjük.
3. Megelőzés. - A 4 szükséges feltétel egyikének meghiúsítása.
   A Coffmanféle 4 feltétel valamelyikére mindig él egy megszorítás.Kölcsönös kizárás. Ha egyetlen erőforrás soha nincs kizárólag 1 folyamathoz rendelve, akkor nincs holtpont se! De ez nehézkes, míg pl. nyomtató használatnál a nyomtató démon megoldja a problémát, de ugyanitt a nyomtató puffer egy lemezterület, itt már kialakulhat holtpont. Ha nem lehet olyan helyzet, hogy erőforrásokat birtokló folyamat további erőforrásra várjon, akkor szintén nincs holtpont. Ezt kétféle módon érhetjük el. Előre kell tudni egy folyamat összes erőforrásigényét. Ha erőforrást akar egy folyamat, először engedje el az összes birtokoltat.
4. Dinamikus elkerülés. - Erőforrások foglalása csak „óvatosan”.
   Van olyan módszer amivel elkerülhetjük a holtpontot? Igen, ha bizonyos info (erőforrás) előre ismert. Bankár algoritmus (Dijkstra,1965) Mint a kisvárosi bankár hitelezési gyakorlata. Biztonságos állapotok, olyan helyzetek, melyekből létezik olyan kezdődő állapotsorozat, melynek eredményeként mindegyik folyamat megkapja a kívánt erőforrásokat és befejeződik.

Bankár algoritmus több erőforrás típus esetén:
Az 1 erőforrás elvet alkalmazzuk:
- Jelölés: F(i,j) az i. folyamat j. erőforrás aktuális foglalása
- M(i,j) az i. folyamat j. erőforrásra még fennálló igénye
- E(j), a rendelkezésre álló összes erőforrás.
- S(j), a rendelkezésre álló szabad erőforrás.

1. Keressünk i sort, hogy M(i,j)<=S(j), ha nincs ilyen akkor holtpont van, mert egy folyamat se tud végigfutni.
2. Az i. folyamat megkap mindent, lefut, majd az erőforrás foglalásait adjuk S(j)-hez
3. Ismételjük 1,2 pontokat míg vagy befejeződnek, vagy holtpontra jutnak.

7.EA

Alapvető memóriakezelés: kétféle algoritmus csoport (swap vagy nincs szükség, ha elegendő a memória)

Monoprogramozás: 1x-re 1 program fut (parancs gépelése, végrehajtása…). Op.rend. alsó címeken, program felette; Alsó címeken a program, felül, ROM-ban oprend.; oprend. alul, felhasználói program ROM-ban

Multi programozás: || több program. Rögzített memóriaszeletek, memóriacsere, virtuális memória, szegmentálás

Rögzített memóriaszeletekkel: beágyazott rendszerek(PIC). Ma több folyamat a memóriában. Memória n szelet, minden szeletre külön várakozási sor, kötegelt rendszereknél tipikus.

Memória felosztás: kihasználatlan partíciók, egy kiürül, sorban következő jön be. OS/360 és OS/MFT rendszernél.

Védelem: OS/MFT PSW (progi állapotszó, 4 bites védelmi kulcs) vagy bázis+határregiszter (lassú)

Memória csere: időosztályos, grafikus felület esetén felejtős. Dinamikus, jobb kihasználtság, de lyukak lesznek, memóriatömörítés szükséges.

Dinamikus memória foglalás: nem ismert egy progi igénye. Kód fix szeletet, az adat és verem változót kapnak csupán.

Dinamikus memória nyilvántartása: allokációs egység definiálása (kicsi=nagy memó igény,kis lukak; nagy=nagy memó veszteség az egységek miatt). Bittérképpel, vagy láncolt listával (külön lyuk és folyamat lista.)

Memóriafoglalási stratégiák:
- First Fit (első hely, ahová befér, leggyorsabb, legegyszerűbb) - Next Fit (előző befektetési ponttól, kevésbé hatékony mint a First Fit) - Best Fit (lassú, sok kis lyuk) - Worst Fit (nem sok luk, nem hatékony) - Quick Fit(méretek szerinti lyuklista, összevonásuk költséges)

Virtuális memória: program a virtuális memóban, több memóriát is használhat, mint a fizikai méret. Monoprogramozásnál is használható

MMU (Memória Menedzsment Unit): virtuális címtér lapokra osztva (tábla). Jelenlét/hiány bit. Figyeli, hogy minden lap a memóriában van-e, ha nem akkor laphiba, majd operációs rendszer lapkeretet kitesz és behozza a lapot. Pl.16bit-4kb lap (lap=2^n, 16 bites virtuális címből 4 a lapszám, a többi offset; 1 jelenlét/hiány bit jelzésére; kimenő 15 fizikai címsínre.) 32 bit esetén 12bit(4kb) lapméret, és 20 bit a laptábla mérete (1MB=1 millió elem) minden folyamathoz saját címtér, laptábla 64 bit esetén ilyen méretű laptábla megvalósíthatatlan
2 szintű laptábla: 10 bit felső szintű és 10 bit második szintű, lapon belüli offset 12 bit
Táblabejegyzés szerkezete: lapkeret száma, jelenlét/hiány bit, védelmi bit (=0:RW, =1:R), dirty bit(módosítás, =1: módosult a lapkeret memória, ki kell írni), Hivatkozás bit =1:hivatkoznak a lapra, Gyorsító tár tiltás bit: (fizikai memó=I/O eszköz adatterület)

TLB (asszociatív memória): MMU-ba kicsi HW egység, kevés bejegyzéssel, szoftveres TLB kezelés, 64 elem miatt a HW megoldás kispórolható

Invertált laptáblák: valós memória méret, laptáblában fizikai memóriából keletkező számú elem, TLB használata, ha hiba, akkor invertált laptáblában keresünk, TLB-be rakjuk.

Lapcserélési algoritmusok: ha nincs virtuális című lap a memóriában, egy lapot kidobni, másikat berakni.
Optimális: címkézés, ahány CPU utasítás hajtódik végre hivatkozás előtt, legkisebb számú lap kidobni (megvalósíthatatlan)
NRU: Modify and Reference bit használata, modify időnként 0-ra, 0-3.osztály: (nem,nem; nem,igen; igen,nem; igen,igen) megfelelő eredmény, nem hatékony
FIFO: legrégibb eldob, ha új kell (előre jön, végéről megy). Javítása a Második lehetőség: ha hiv.bit 1→sor eleje bit=0
Óra: Második Lehetőséghez hasonló, mutatóval járjunk körbe, legrégebbi lapra mutat, ha hiv.bit=1→0, továbblépünk
LRU: legrégibb algoritmus. HW vagy SW megvalósítás.
NFU: lapokhoz számláló, ehhez referencia bitet adunk, óramegszakításkor, legkisebb értékűt dobjuk el, nem felejt!

Munkahalmaz modell: előlapozás, használtak a fizikai memóriában tartva, lapnyilvántartás (Óra javítása: WSClock)

Lokális, globális helyfoglalás: laphibánál hogyan vizsgáljuk, méret szerinti lap-dinamikus, PFF alg. laphiba/mp (teherelosztás)

Helyes lapméret meghatározása: kicsi (lapveszteség kicsi, nagy laptábla), nagy (fordítva) n*512 bájt a lapméret

Szegmentálás: virtuális memó (1D címtér, 0-tól maxig), több progi dinamikus területtel, egymástól független címtér (szegmens), cím 2 része (szegmens szám, ezen belüli cím), egyszerű osztott könyvtárak, logikailag tagolható (adat és kód), védelmi szint egy szegmensre, fix lapméret, változó szegmensméret

Pentium processzor virtuális címkezelése:
sok szegmens: 2^32 bite (4 GB), 16000 LDT (folyamatonként) GDT(Globbal Descriptor Table 1db) fizikai cím: szelektor+offset művelet szegmens elérés: 16 bites szegmens szelektor Lineáris cím TLB a gyors lapkeret eléréshez védelmi szintek: 0-3 (Kernel,Rendszerhívások, Osztott könyvtárak, Felhasználói programok) alacsonyabbról adatelérés engedélyezett, fordítva tiltott eljárás hívása ellenőrzött módon felhasználói programok osztott könyvtárak adatait elérhetik, de nem módosíthatják.

Fontosak még: