Operációs Rendszerek Szóbeli

Tartalom

1. TÉTEL 3

Operációs rendszerek feladatai: 3
Operációs rendszerek generációi: 3
Operációs rendszerek csoportosításai: 4
Kernel struktúrák: 4
Erőforrás típusok: 4
Tipikus problémák: 4

2. TÉTEL 5

Operációs rendszerek felületei (programozók és felhasználók felé): 5
Kernel API: 5
Rendszerhívási osztályok: 5
A „burok” (shell) kifejezés kettős értelme: 5
Csatorna átirányítás: 6
Parancs behelyettesítés: 6
Környezeti változók és tárolása, láthatósága, öröklése: 6
Fájl minta illeszkedés: 6
Karakter semlegesítés: 6

3. TÉTEL 7

Vezérlési szerkezetek a burok programnyelvben (shell-ben): 7
segédprogramok és szűrők: test, expr, read, cut, head, tail, grep: 7
Reguláris kifejezések: illeszkedés, védés, speciális karakterek, semlegesítés: 8
Az awk programozása: 9

4. TÉTEL 10

Fájlrendszerekkel kapcsolatos elvárások: 10
a FAT fájlrendszer felépítése, szabad helyek kezelése, blokkok fejléce, adatterületek tartalma könyvtárak és fájlok esetében, példa egy fájl beolvasására: 10
az UFS fájlrendszer felépítése, szabad helyek kezelése, blokkok fejléce, példa egy fájl beolvasására: 11
fájlrendszer felcsatolása („mount”-olás) Unix rendszerben: 12
az /etc/fstab és /etc/mtab fájlok: 12
a /dev könyvtár: 12
loopback device, soft és hard linkek: 12
FUSE, VFS, NFS: 12

5. TÉTEL 14

A Linux boot lépései 14
futásszintek 14
IP konfigurálás Unix alatt: IP cím hozzárendelés, netmask, routing, gateway, DNS 14
portok és szolgáltatások 15
DHCP, CIFS/Samba, CUPS, SSH, X11, XDMCP, vékony kliens koncepció. 15

6. TÉTEL 16

Multiprogramozás és lehetőségei, látszat párhuzamosság, megvalósítások 16
processzek alapfogalmai (Process Control Blocks, Process Image) 16
processz állapotok (futó, futásra kész, blokkolt, felfüggesztett) 16
processz állapot-átmenet gráfok 17
processz váltás lépései 17
Szálak (thread-ek) és használatuk (POSIX rendszerben). 17

7. TÉTEL **18**

Processzek közti kommunikáció (IPC) 18
Kommunikációs alapfogalmak: blokkolás, szinkron, címzés, számosság, szimmetria, közvetettség, puffer kapacitás 18
primitív mechanizmusok (csatorna, medence) 18
példák (signálok, környezeti változók, fájlokon keresztüli kommunikáció, stb.) 19
osztott memória 19
üzenetsor. 20

8. TÉTEL **21**

Időkiosztás (scheduling), ütemezési stratégiák 21
CPU ütemezés eszközei 21
Ütemezési döntési helyzetek 21
Prioritást befolyásoló tényezők 22
Processzek életszakaszai 22
Ütemezési algoritmusok: FCFS, SJF, Policy Driven Scheduling, RR, Multilevel Feedback Queue Scheduling. 22
***Multilevel Feedback Queue Scheduling**: Több összetevőtől függ a sorrend kialakítása. Több ready sor van különbözőprioritási szintekhez. Elvileg mehet minden sorhoz más scheduling algoritmus.* 22
öregedési algoritmus 22
példák (Vax, Unix, Linux, NT). 23

9. TÉTEL **24**

Versenyhelyzetek: Konkurens folyamatok közötti kommunikáció, versenyhelyzet, kritikus szekció fogalma, kritikus szekció sikeres megvalósításának a feltételei. Konkurrens programozás alapjai: megszakítások tiltása, zárolásváltozó, szigorú alternáció, Peterson módszere, Test and SetLock utasítás 24
prioritás inverzió 25
gyártó- fogyasztó probléma 25
szemaforok, mutexek, monitorok 25
5 filozófus problémája, író- olvasó probléma, alvó-borbély problémája. 25

10. TÉTEL **26**

Holtpont definíciója. 26
Holtpont kialakulásának feltételei (Coffman) 26
Példa holtpont kialakulására 26
Holtpont feloldási stratégiák (struccpolitika, felismerés és helyreállítás, megelőzés, dinamikus megoldás) 26
példák. 27
Bankár algoritmus 27
Dijkstra algoritmus. 27

1. Tétel

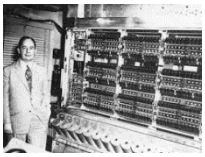
Operációs rendszerek feladatai (Extended Virtual Machine, Resource Mananagment, Responding Machine); operációs rendszerek generációi; operációs rendszerek csoportosításai; kernel struktúrák; erőforrás típusok; tipikus problémák.

Operációs rendszerek feladatai:

1. **Extended Virtual Machine:** Kiterjesztett virtuális gép, a perifériák átfogó ismeretének elkerülhetővé tétele. A különbözőségeket szimbolikus nevek mögé rejti és standard interface-t biztosít. A virtuális gépet könyebb programozni mint az alatta létező hardvert. Ebből aszempontbol kényelmessé teszi a hardver használatát.
2. **Resource Mananagment:** Erőforrás kezelés, hardware-ek és software-ek kezelése. Feladata a processzek memória igényének kielégítése és processzel egymástól való eredménye.
3. **Responding Machine:** Válaszló gép, Kernel API szolgáltatások: Fájlrendszerhez (fopen,fclose), process kezeléshez(fork,waitpid), időzítések (time, delay). Minden Kernel API hívás egy Kernel Trap-et eredményez.

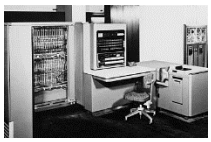
Operációs rendszerek generációi:

1. **Generáció: 1945-55 (vákumcsövek és kapcsolótáblák)**



- Elektroncsöves, dug táblák, lyukkártyák
- Gépi nyelvű programozás
- Nincs még operációs rendszer
- A tervező, megépítő, karbantartó ugyan az a személy.
- Neumann, Zuse

2. **Generáció: 1955-65 (tranzisztorok és kötegelt rendszerek)**

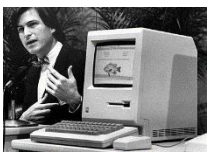


- Karakter orientált és szószervezésű gépek.
- Külön építő, karbantartó, programozó és felhasználója van
- JOB fogalom megjelenése: a munka egysége. amit a kezelő az operációs.rendszernek ad. **Betölt-Lefordít-Betölt-Végrehajt** ciklus (load-translate-load-exec).

3. **Generáció: 1965-80 (integrált áramkörök és multiprogramozás)**

- IC-k megjelenése, multi programming (egy időben több program fut).
- Szó orientált, idő osztásos, memória praticionálás, spooling (folyamatos periféria feladatok végrehajtása a szálon).
- software kompatibilitás fogalma
- C nyelv megjelenése.

4. **Generáció: 1980-tól napjainkig (személyi számítógépek)**



- Grafikus felhasználói területek megjelenése (GUI).
- LSI, VLSI : lényege, hogy több tranzisztort egyesítenek, hogy egy chipet alkosson.
- Hálózati operációs rendszerek, teljes párhuzamosság
- Objektum orientált programozás
- A mai OS-ek kevésbé architektúra függőek.
- Moduláris kernelek, linux, windows, osx a fő operációs rendszer.
- Mindneki felhasználó

Operációs rendszerek csoportosításai:

1. **Célkitűzés szerint:**
    - **Mainframe**
    - **Server/desktop**
    - **Single/Multi-processor**
    - **Real Time**: Valós időben végzi el a feladatokat, prioritások használatával és megszakításokkal. Multimédiában és biztonsági rendszereknél alkalmazzák.
    - **Embedded:** egy konkrét feladat ellátására tervezték meg.
2. **Hardver függőség szerint:** Személyi, kis és nagy gépek operációs rendszere. Architektúra független.
3. **Cél szerint:** Általános vagy speciális célú.
4. **Processzek, processzorok, felhasználók szerint:**
    - single vagy multi tasking
    - single vagy multi user
    - megosztott
5. **Időkiosztás szerint:** Szekvenciális, kooperatív, vagy beavatkozó, real-time.
6. **Memória kezelésszerint**: **Valós** vagy virtuális címzésű.

Kernel struktúrák:

1. **Monolitikus:** Induláskor betöltődik a teljes kernel, ami meglehetősen sok feladatot lát el. A komponensek között nincsenek alá-fölé renledési viszonyok és hatásköri kérdések.
2. **Mikro-kernel (Moduláris):** Egy kicsi mag, minden egyéb userspace-en, de legalábbis szeparált területen fut. Biztonságos a fejlesztése, mert egy userpace hiba nem rántja magával a teljes os-t, viszont a sok kernel trap drága művelet ezért lassú.(Windows NT.)
3. **Hibrid:** Előző kettő kombinációja. A hybrid kernelek alapjában véve olyan mikrokernelek, amelyekben kevesebb absztrakciót használva, gyosabban fusson.(Linux Kernel).

Erőforrás típusok:

1. **memória:** szabad terület managelése
2. **disk:** filerendszer
3. **periféria:** sorban állások, prioritások
4. **cpu, DMA:** időosztás prioritás

Tipikus problémák:

1. **Versenyhelyzet(race condition):** A párhuzamos program lefutása során a közös erőforrás helytelen használata miatt a közös erőforrás nem megfelelő állapotba kerül.
2. **Holtpont:** A közös erőforrások hibás beállítása vagy használata miatt a rendszerben a részfeladatok egymásra várnak.
3. **Livelock:** Többnyire a hibás holtpont feloldás eredménye.
4. **Prioritás inverzió:** Prioritásos ütemezőkben fordulhat elő, de az erőforrás használattal is összefügg.

2. Tétel

Operációs rendszerek felületei (programozók és felhasználók felé); Kernel API; rendszerhívási osztályok; a „burok” (shell) kifejezés kettős értelme; parancs, cső, parancs lista;csatorna átirányítás; parancs behelyettesítés; környezeti változók és tárolása, láthatósága, öröklése; fájl minta illeszkedés; karakter semlegesítés.

Operációs rendszerek felületei (programozók és felhasználók felé):

1. **programozók felülete:** Alkalmazásokból és rendszer processzekből rendszerhívások és eseménykezelők: alkalmazás-programozási felület azaz API.
2. **felhasználók felülete:** Parancsnyelvi vagy grafikus felület (burok,GUI), Segédprogramok készlete.

Kernel API:

- Engedélyezi a userspace-ben lévő programokat, hogy hozzáférhessenek a rendszer erőforrásaihoz a os kernelben. Ezt a rendszerhívások segítségével valósítja meg.
- Rendszerhívások: rutin hívás, röviden. Van benne egy kicsomagoló rész az argumentumok kezelésére, továbbá ellenőrzött módváltó instrukció folyam valamint call jellegű belépés a kernel diszpécserébe.

Rendszerhívási osztályok:

- Processz menedzsment:
  - create, terminate process
  - load, execute
  - end , abort
- Informálódó beállító: get, set time , date
- Fájlok, jegyzékek:
  - create, delete files
  - open, close file
  - read, write, reposition
- Eszközmanipulációk:
  - request, relase devic
  - read, write, reposition
  - get,set device attributes

A „burok” (shell) kifejezés kettős értelme:

A burok egyrészt parancsértelmező process, másrészt egy programnyelv. Mint process a /bin /sh betölthető, futtatható program egy futási példánya. Programnyelvként egy szövegfájl, ami parancsokat (input) sorokat tartalmaz és odaadható a burok processznek, hogy futtassa le.

Parancs, cső, parancs lista:

1. **Parancs:** Fehér karakterekkel határolt szavak sora, ahol az első szó a parancs neve a többi szó pedig az argumentumok. A burok beolvassa, értelmezi és végrehajtja. **echo “Parancs”**
2. **Cső:** Parancsok sora | operátorral elválasztva. A baloldali parancs kimenete meghatározza a job oldali kimenetét. A cső visszatérési értéke a jobboldali parancsé.
3. **Parancs lista:** Csővezetékek sora lista operátorral összekötve. Listaoperátor lehet \n soros végrehajtása a csöveknek. & aszinkron végrehajtás. && folytatja a listát ha csőbal normális visszatérésű. || folytatja a listát ha csőbal nem normális visszatérésű. A lista visszatérési értéke a parancs visszatérési értéke.

Csatorna átirányítás:

- >,>>,< operátorokkal oldható meg.
- Mielőtt végrehajtódik a parancs a shell nézi, hogy van-e bármelyik operator a parancsban.
- Ha talál, akkor szeparált processzeket készít, és azokba a file-okba képzi le az adatfolyamot vagy azokból a file-okból képzi le, majd abban hajtják végre a listát/parancsot.

Parancs behelyettesítés:

ALT Gr+7 közé zárt parancs lefut és kimenete kifejtődik. Nem egy értéket adunk meg hanem egy parancsot aminek az eredménye less a változó értéke `$sordb=`wc -l teszt.txt``

Környezeti változók és tárolása, láthatósága, öröklése:

Minden processzenek így a buroknak is van környezete. A környezeti változók, a fontos információkat tartalmazó szöveges szimbólumok, amelyeket az operációs rendszer tárol. A környezeti változók a programokon kívül léteznek, de a programok lekérdezhetik és felhasználhatják működésükhöz őket.
A gyermek process a környezetet örökli a szülőtöl.

- **Többszintű öröklödés:** `$ export HOME PATH` , Exportálással teszünk változót a környezetbe. Exportálással cask a leszármazottak örökölhetnek.
- **Egyszintű öröklődés:**
  a. Egyszintes környezetbe tétel
  b. ha a parancs külső akkor látni fogja, ha belső akkor nem. `$ valt = kif parancs`
  c. ha a parancs külső akkor nem látja , ha belső akkor igen. `$ valt = kif; parancs`

Fájl minta illeszkedés:

Karakterek beillesztésének a segítségével tudjuk kezelni. Szokásos karakter önmagára ilelszkedik. A **?** egyetlen bármely karakterre illeszkedik `ls f?le`. A ***** tetszőleges számú karakterre illeszkedik `ls file*`.. A **[]** egyetlen, valamely bezárt karakterre illeszkedik `ls f[oi]le`. Egyetlen, a zárójeleben szereplő karaktereket kivéve illeszkedik `ls f[!oi]le`.

Karakter semlegesítés:

A legegyszerűbb mód ha **('…')** közé zárva irjuk az adott metakarektereket amelyeket védeni szeretnénk a shell interpretálásától. Karakterenként \ idézőjel közé zárva **" *"**.

3. Tétel

Vezérlési szerkezetek a burok programnyelvben (shell-ben); segédprogramok és szűrők: test, expr, read, cut, head, tail, grep; Reguláris kifejezések: illeszkedés, védés, speciális karakterek, semlegesítés; Az awk programozása

Vezérlési szerkezetek a burok programnyelvben (shell-ben):

1. Ide tartoznak a szekvenciális programszerkezetek, azaz a **parancslisták**, az **elágazások** és a hurok (**loop**).
   a. **Elágazások:**

if,then,elif,else,fi

```
if [ $1 = “majom”]; then
echo "Ez egy majom!";
else
echo "Ez nem az!"
fi
```

case

```
case majmok in
  1 )
     echo "majom01" ;;
  2 )
     echo "majom02" ;;
esac
```

   b. **Hurkok**:
   - **Forciklus**: for valt [in szolista] do plista done. A változó render felveszi a szólista elemeit és minden értékével végrehajtódik a plista.
   - **While ciklus**: while plista1 do plista2 done. Végrehajtódik a plista1 és ha normális a visszatérése azaz igazat ad akkor végrehajtódik plista2, addig amég plista1 nem ad vissza másmilyen értéket.

segédprogramok és szűrők: test, expr, read, cut, head, tail, grep:

A **segédprogram** egy szoftver, amely segít ellátni a felhasználó, vagy a rendszergazda egy konkrét számítógéppel kapcsolatos feladatait.A segédprogramok jelentős része az operációs rendszerek szolgáltatásaihoz tartoznak, ahol minden segédprogram egy pontosan meghatározott feladat végrehajtását segíti.

1. **test parancs:** Két szintaktisa létezik: test kifejezés vagy test [kifejezés]. Fájlokkal kapcsolatos tesztek vagy adatszerkezetek relációi lehetnek.:

test -f file	text -d file	text -r file	text -s file
file létezik és sima file	file létezik és jegyzék	file létezik és olvasható	file létezik és 0-nál hosszabb

2. **adatrelációk:**

[a -gt b]	[a -lt b]	[a -eq b]	[a = b]
a nagyobb mint b	a kisebb mint b	értékek egyenlőek	szövegként egyenlőek

3. **expr parancs:** kiértékelődik a kifejezés és az eredmény az atdout-ra írodik, amit az echo jelez.Algebrai operátorokat használ (+,-,/,*) amelyeket semlegesíteni kell pl.: `valt=`expr $valt+2``;
4. **read:** Beolvas egy bemeneti értéket az stdin-ről és az első szót a valt1-be a másodikat pedig a valt2-be másolja: `read valt1 valt2` Ha több aszó mint a változó a maradékot az utolsóba teszi bele.
5. **cut:** Ez egy szűrő, beolvassák a bemenetet majd egy szűrési műveletet követve a szűrt eredmény jelenik meg a kimeneten. pl `cut -d -f1 -2 /etc/passwd`. Kivágja az első 2 oszlopot a passwd-ből. -c oszlopokat -f mezőket tudunk kivágni.

6. **head:** Kiirja egy szöveges állomány vagy egy csővezeték eredményének első néhány sorát. Alapértelmezetten 10-et de növelhető a szám. **head -n 5 fájl** kiirja a fájl első 5 sorát.
7. **tail:** Kiirja egy szöveges állomány vagy egy csővezeték eredményének utolsó néhány sorát. Alapértelmezetten 10-et de növelhető a szám. **tail -n 5 fájl** kiirja a fájl utolsó 5 sorát.
8. **grep**: fájllista fájlinak a sorait olvassa, és minden sorra illeszt egy mintát, ha találat van akkor kiírja azt a sort.Visszatérési érték lehet 0 (van találat), 1 (nincs találat), 2 (akadálybaütközik/hiba) Opciók:
   -n: sorszámok kiírása is
   -v: nem egyezést keres
   -y: kis/nagy betűket megkülönbözteti
   -c: csak sorszámokat ír ki
   grep szo szoveg.txt: kiírja a sorokat amiben szerepel a „szo”

Reguláris kifejezések: illeszkedés, védés, speciális karakterek, semlegesítés:

A **reguláris kifejezés** egy olyan, bizonyos szintaktikai szabályok szerint leírt string, amivel meghatározható stringek egy halmaza. Nevének röviditésére gyakran a **regexp** vagy **regex** kifejezést használatos.:

Atom: kerek zárójelek közötti regex vagy:

c	\c	.	[chars]	[^chars]	^	$
normális karakter önmagára ilelszkedik	**speciális karakter, önmagára illeszkedik**	**bármely nem új sor karakterre ileszkedik**	**egyetlen a bezártak közül bármelyikre**	**egyetlen a bezártakon kivül**	**első karakter előtti üres szövegre**	**utolsó karakter utáni üres szövegre**

Lezárt: Olyan atom amit opcionálisan egy postfix operator követ.

atom*	atom+	atom-
atomnak 0 vagy több előfordulására illeszkedik	atomnak 1 vagy több előfordulására illeszkedik	atomnak 0 vagy 1 előfordulására illeszkedik

Összefűzött: Akárhány egymásutáni lezártból állhat, először az első majd a második lezárt illesztésekor az összefűzött is illeszkedik.

Unio: Akárhány összefűzött a | karakterrel (**Alt Gr + W**) elválasztva. Akár az első akár a második összefűzött illeszkedik akkor az unio illeszkedik.

Illeszkedés:

abcd	^string	string$	[..],[^..]	[a-z][0-9]	[^A-D0-2]	c+	^$
a karakterek önmagukra illeszkednek	az összefűzött string a sor elején illeszkedik	az összefűzött string a sor végén illeszkedik	atom	atom egyetlen bezárt karakterre a tartományból	több tartomány és negáció is lehet	lezárt, 1 vagy több előfordulása a c-nek	összefűzött, cask üres sorra illeszkedik

Az awk programozása:

Mintakereső és feldolgozó. Szövegfolyam sorokat olvas, minden sorban mintákat keres és a mintához tartozó akciókat végrehajtja.Az awk adatvezérelt nyelv, minták és tevékenységek sorozatából áll. Beolvas egy sort, automatikusan mezőkre (szavakra) darabolja a megadott feltételek szerint elvégzi a minta egyezőségének vizsgálatát. Ha van egyezés, akkor a tevékenységet végrehajtja majd aszabványos kimenetre írja. Következő részekből épül fel egy awk program:

1. előkészítő rész: **BEGIN** parancs
2. **saját függvénydefiníciók**
3. **programtörzs**: /”1. minta”/{„parancsok”}
4. befejező rész: **END** parancs

Az awk sok nyelvi elemet használ a programozás megkönnyítésére pl rekordok, tömbök, minták, tevékenységek, operátorok, be és kimeneti utasítások, konstansok, saját függvények stb..

4. Tétel

Fájlrendszerekkel kapcsolatos elvárások; a FAT fájlrendszer felépítése, szabad helyek kezelése, blokkok fejléce, adatterületek tartalma könyvtárak és fájlok esetében, példa egy fájl beolvasására; az UFS fájlrendszer felépítése, szabad helyek kezelése, blokkok fejléce, példa egy fájl beolvasására;fájlrendszer felcsatolása („mount”-olás) Unix rendszerben; az /etc/fstab és /etc/mtab fájlok; a /dev könyvtár; loopback device, soft és hard linkek; FUSE, VFS, NFS;

Fájlrendszerekkel kapcsolatos elvárások:

A számítástechnika egy fájrendszer alatt a számítógépes fájlok tárolásának és rendszerezésének a módszerét érti, ide értve a tárolt adatokhoz való hozzáférést és az adatok egyszerű megtalálását is.

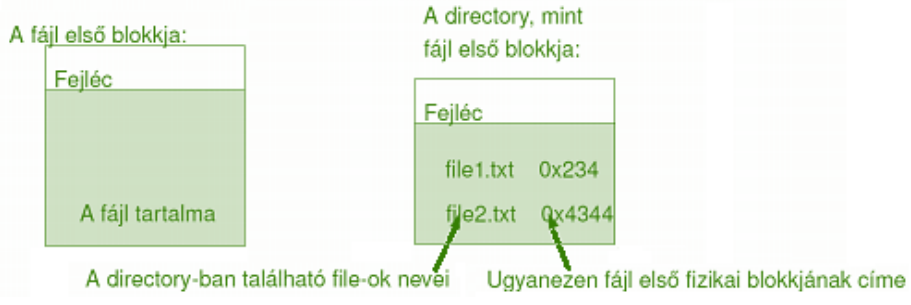
Elsődleges elvárások:	**Másodlagos elvárások:**
• **namespace**: nevekkel lehessen hivatkozni az adatokra (fájl nevek) • **szabad terület** és **foglalat terület managmentje**, azaz tudjuk, hogy mere vannak az adatok a tárolón • **transzparencia**: minden hardware-t ugyanúgy lehessen kezelni azaz gyártótol mentes legyen egy hardware kezelése.	• **tools**: eszközök legyenek hozzá formázásra • **robosztusság**: ha elmegy a táp ne korruptálódjon azaz ne boruljon a sorrendiség a nevek ne romolhassanak el • **Hozzáférés** és jogosultság szabályzása • **fragmentation policy**: az adatok fregmentációjának csökkentése • **cache**: legalább olvasásra

a FAT fájlrendszer felépítése, szabad helyek kezelése, blokkok fejléce, adatterületek tartalma könyvtárak és fájlok esetében, példa egy fájl beolvasására:

A **FAT fájrendszer** a **Windows NT** operációs rendszer által támogatott fájrendszerek legegyszerűbbike, legfőbb sajátosságaa fájlkiosztási tábla FAT, ami a kötet legfelső szintjén elhelyezkedő adattáblázat.A kötet védelme érdekében a fájlkiosztási táblának két példánya van, ha az egyik megsérül a másik segítségül hívható. FAT-ra formázott kötetek alkotóegységei a klaszterek (clusters). A klaszter méretét a kötet méretehatározza meg 512 bájt és 64 KB közötti lehet. Megkülönböztethetünk FAT12,FAT16 és FAT32-t. A FAT-ek után szereplő számok a bitekre utalnak amelyekkel azonosítja a klasztereket, így minél nagyobb bit annál több klasztert tud azonosítani és így felhasználni. Manapság már csak a 32-est használják. A FAT tábla tartalmazza azokat az adatokat, hogy egy fájl tartalmának végigolvasásához melyik klaszter után melyik klasztert kell olvasni.

Szabad helyek kezelése: A fájl egy folytonos fizikai blokk sorozatot foglal el.Hozzáférés egyszerű és gyors HDD esetén (a sáv folytonosan olvasható a forgó lemezről).Növekvő méretű fájloknak a helyfoglalás problémás.Milyen méretű szabad helyet allokáljunk?Új fájlok számára megfelelő szabad hely megtalálása nehéz, külső tördelődés lép fel.Fájl törlése után a méretének megfelelő számú blokk felszabadul.
Erre a helyre kisebb vagy egyenlő méretű fájl írható. Ugyan azokat az algoritmusokat használhatjuk, mint a memória foglalás során (First fit,Next fit, Best fit, Worst fit).Növekvő fájlok esetén a Best fit allokációs stratégia különösen veszélyes.A fájl nagyobb szabad helyre másolása (erőforrás igényes).

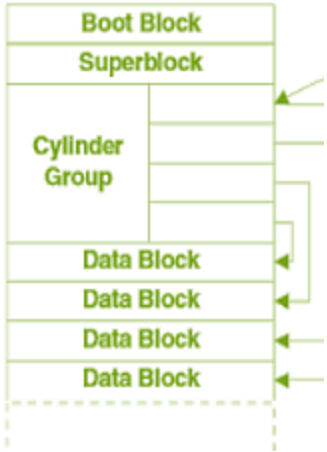
Blokkok fejléce:



az UFS fájlrendszer felépítése, szabad helyek kezelése, blokkok fejléce, példa egy fájl beolvasására:

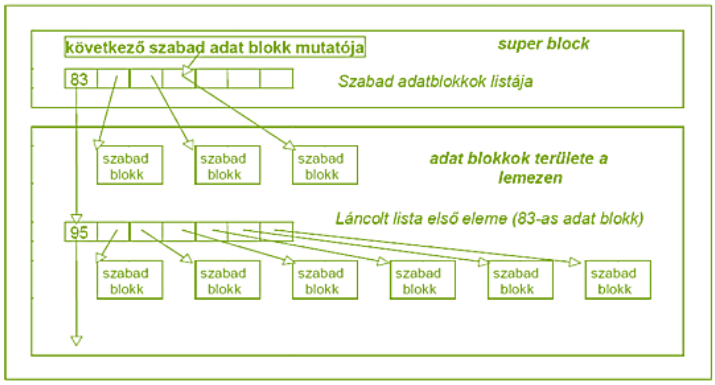
Az UFS fájlrendszert 1982-ben vezették be. Sok Unix és Unix-alapú operációs rendszer használta. A maximális fájlméret 273 bájt (8ZiB) l ehet. A maximális fájlnév-hosszúság 255 bájt.

az UFS fájlrendszer felépítése:



1. **Boot szektor:** a paríció elején található pár blokk, amit a fájlrendszerből külön kell inicializálni.
2. **szuperblokk:** ez tartalmazza a „magic number" és más alapvető számokat, amik leírják a fájlrendszer geometriai, statisztikai és viselkedésbeli finombeállításait.
3. **Cilinder csoportok:** Minden cilindercsoport tartalmazza:a szuperblokk biztonsági másolatát a cilindercsoport fejrészét i-node-ok számát (mindegyik tartalmazza a fájl attribútumait) adatblokkok számát.
4. **i-node-ok:** fájl típusa (speciáls, adat, könyvtár bejegyzés, PIPE/FIFO) a linkek számát, amiko az adott inode-ra mutatnak az eszköz és az i-node ID-t fájlelérési jogosultságokat mód flageket a fájlhoz tartozó adatblokkok elérési információját.

Szabad helyek kezelése: Két tárolási korlát figyelhető meg:



1. **fájrendszer méretére vonatkozó** (fizikai méretnél nem lehet nagyobb mint a logikai)
2. A **szabad adatblockkok tárolása** szabad adatblokkokra mutat tömbök láncolt listájával történik.

Példa egy fájl beolvasására:

1. fopen (nyitás módja , fájl neve)
2. nyitáskor az eltolási érték 0
3. jogok ellenőrzése megtörténik
4. új bejegyzés a process FDT-ben
5. read(olvasandó byte szám)
6. chown uid megváltoztatása
7. utimes időpontok módosítása

fájlrendszer felcsatolása („mount”-olás) Unix rendszerben:

Használatba vétel előtt a fájlrendszert csatolni kell a rendszerünkhöz. Ehhez bekel olvasn ia szuperblokkot és a fájrendszer táblában egy új bejegyzést kell létrehozni. Csatolás után a fájlrendszer fájlai elérhetővé válnak.

az /etc/fstab és /etc/mtab fájlok:

fstab= file system table

Az /etc/fstab szerepe UNIX rendszerben , hogy az itt lefektetett szabályok megkönnyítik az állományrendszerek használatát. Megléte rendkívül fontos, mivel az fsck és a mount (tehát a boot során az automata mount is) ebből tájékozódik az elérhető állományrendszerekről. Az fstab kilistázza az elérhtő lemez partíciókat és egyéb típusú file rendszereket az olyan adat forrásokat, amik nem lemez elhelyezkedésűek.

fs_specs	fs_file	fs_vfstype	fs_mntops
fizikailag hol van a kezelendő fájlrendszer	hová kerüljön montolásra a fájlrendszer	állományrendszer típusa	mountolási beállításokat tárolja

mtab= mounted file system table. Az mtab file egy rendszer információs file. Kilistázza az összes jelenlegi mountolt file rendszert a beállítási opciókkal együtt.

a /dev könyvtár:

dev= device file, két részből áll az egyik azonosítja a kontrollert, a másik a tényleges eszközt és drivert. A dev könyvtár a speciális eszközfájlokat tartalmazza az összes eszköz számára. Az eszközfájlok a telepítéskor jönnek létre, illetve később a /**dev/MAKEDEV** szkript segítségével. **A /dev/MAKEDEV.local** egy szkript, melyet a rendszeradminisztrátor ír. példák:

/dev/dsp	/dev/fd0	/dev/loop0	/dev/null	/dev/md0	/dev/pt0
hangkártya és a hangot létrehozó program	első hajlékonylemez meghajtó	első loopback eszköz	ami ide kerül az megsemmisül	RAID eszköz számára	párhuzamos potra csatlakoztatott szalagos eszköz.

loopback device, soft és hard linkek:

Loopback device: Egy olyan mechanizmus mely a file-okat úgy értelmezi,mintha igazi eszközök lennének. Az előnye ennek, hogy minden igazi lemezentárolt eszközt tudunk használni a loopback segítségével. KB mint egy image file.

Soft link: A soft link (symbolic link (symlink)) egy speciális fájltípust jelöl,amely valójában egy hivatkozás egy másik fájlra vagy könyvtárra. A soft linkeknem közvetlenül mutatnak adatra, hanem egy elérési útvonalat tartalmaznak, amelyből a rendszer képes egy hard linket (vagy egy másik soft linket) beazonosítani.

A hard link: rögzített hivatkozás egy hivatkozás vagy mutató egy adattároló eszközön elérhető adatra. A legtöbb fájlrendszerben az elnevezett fájlok hard linkek. Hard linkkel csak azonos fájlrendszerben létező adatra lehet hivatkozni.

FUSE, VFS, NFS:

FUSE: A FUSE (**Filesystem in USErspace**) segítségével képesek vagyunk felhasználói térben filerendszert megvalósítani. A FUSE kommunikációsfelülete egyszerű, hatékony, biztonságos és emellett támogatja a szokásosfilerendszer szemantikákat.

VFS: Virtual File System. Egy szoftverben megvalósított absztrakció, aminek lényege hogy filerendszerként tekintünk valamit, ami valójában klasszikus értelemben nem az (pl. FTP szerverek listája vagy egy .ZIP file tartalma), vagy pedig több fizikailag különböző filerendszer eléréséhez biztosítunk egy API-t, amit szintén szokás VFS-nek nevezni.

NFS: Network File System. A SUN által kitalált hálózati filerendszer, mely erősen támaszkodik az távoli hozzáférésre. Lehetővé teszi hálózaton keresztül adatok megosztását. (**UNIX**, más oprendszereknél nem elterjedt).

5. Tétel

A Linux boot lépései; futásszintek; IP konfigurálás Unix alatt: IP cím hozzárendelés, netmask, routing, gateway, DNS; portok és szolgáltatások; DHCP, CIFS/Samba, CUPS, SSH, X11, XDMCP, vékony kliens koncepció.

A Linux boot lépései

A Master Boot Record (MBR) vagy más néven a partíciós szektor a merevlemez legelső szektorának elnevezése. Az MPR-t betöltő kódnak vagy rendszerindító kódnak is nevezik. Lépések:

1. **kernel image betöltése:** elindítja a különböző rendszerfunkciókat (hardware és memória lapozás)
2. **start kernel:** elvégzi arendszerbeállítások nagyrészét (megszakítások, memória managelés és driver inícializáló)
3. **inittab végrehajtása**: kernel betöltése és elindulása után betölti a megadott ramdisk-et és elkezdi olvasni az inittabot.
4. **rc.d végigolvasása majd végrehajtása:** végső beállításokat állítja be mint a login, nyomtató és a futásszintek.

futásszintek

0-s futásszint: Halt System: minden folyamat leáll, a fájlrendszereket leválasztják, a felhasználókat kijelentkeztetik, a gép biztonságosan kikapcsolható
1-es futásszint: Single user mode, rendszergazdaként egy felhasználós mód, minden /etc/fstab állományrendszer felcsatolásra kerül
2-es futásszint: Alap több felhasználós mód, NFS használat
3-as futásszint: Teljes több felhasználós mód, parancssoros használat
4-es futásszint: használatlan
5-ös futásszint: Több felhasználós mód, GUI használat
6-os futásszint: Rendszer újraindítás (REBOOT)

IP konfigurálás Unix alatt: IP cím hozzárendelés, netmask, routing, gateway, DNS

IP címet Linux alatt az **ifconfig** paranccsal kérhetünk le és az alábbi információk jelenhetnek meg.

IP címet a következő paranccsal tudunk hozzárendelni egy interface-hez:

ifconfig eth0 192.168.1.1 netmask 255.255.255.0 Az **eth0 interface**-nek állítottunk be ezzel egy statikus IP címet egy maszkkal.

netmask: A hálózaton nem csak a host-okat (számítógépeket, munkaállomásokat) azonosítják IP címmel, hanem a hálózatokat is. Egy host neve úgy áll össze, hogy a cím elejénlevõ bitek határozzák meg a hálózat címét, a fennmaradó bitek pedig a host-ot azonosítják a hálózaton belül. Az, hogy melyik rész a network cím, és melyik azonosítja a host-ot a netmask határozza meg.

routing: Az útválasztás, hálózati forgalomirányítás (hálózati forgalom, telefonhálózatok, elektronikus adathálózat).

gateway: A gateway szerepe, hogy a nem lokális IP címekre eljuttassa, továbbadja más hálózatokba az IP csomagokat. A lokális vagy nem lokális a netmask-ból és az IP-ből látszik.

DNS: DNS feladata az IP címek és nevek összerendeléseinek a feldolgozása.

portok és szolgáltatások

Az IP egy gépet azonosít, de a szolgáltatások további belső azonosítókkal bírnak, amelyeket portoknak nevezünk.

- **22**: **SSH** távoli bejelentkező protocoll
- **80**: **www**, http internet protokoll
- **110**: **pop3**, üzenetküldő protokoll
- **25**: **SMTP**: email protokoll
- **53**: **DNS**
- **143**: **IMAP4**, email fogadás
- 443: **HTTPS**
- **502**: **Modbus**

DHCP, CIFS/Samba, CUPS, SSH, X11, XDMCP, vékony kliens koncepció.

A **DHCP** segítségével megadhatjuk, hogy minden kliens gépnek dinamikusan azaz automatikusan és nem manuálisan beállított IP címet osszon ki a server, valamint konfigurálja a gatewayt és a DNS-ét.

CIFS/SAMBA: A Samba egy programgyűjtemény, mely megvalósítja a Server Message Block (röviden: SMB) protokollt UNIX rendszereken.(Fájl és nyomtatási szolgáltatások, Hitelesítések és engedélyek, Névfeloldás, Tállózás)

CUPS:UNIX gépeknél valósítja meg, hogy nyomtató szerverekként működjenek. Egy olyan gép ami CUPS-ot futtat képes fogadni nyomtatási feladatokat a kliensektől, majd feldolgozni ezeket és továbbküldeni a megfelelő nyomtatóhoz.

SSH: Titkosított csatornát biztosít két gép között. Két komponensből áll az egyik a kliensen van, mely biztosítja a terminal-t, ez az ssh-client, a másik pedig a szerveren ez az sshd.

X11: Ez egy protokoll melyet akár hálózaton keresztül is használhatunk és biztosítja az alkalmazások számára, hogy grafikus elemeket használjon.

XDMCP: X11-es szabványt használva lehet csatlakozni távoli gépekhez, de a szervernek futtatnia kell egy X server display manager Control Protocolt is.

Vékony kliens koncepció: Néhány erős szerver és néhány régi vagy elavult számítógéppel (thin client) is ki lehet elégíteni a felhasználók igényeit. A lényege, hogy nincs winchester-e hanem a hálózatból bootol be (tftp), csupán egy X server futtatására képes CPU, memória és VGA található benne, esetlegegy USB port.Legfeljebb fele annyiba kerül mint egy átlag PC. A feladatokhoz szükséges erőforrásokat a serverből nyeri, ott végzi el a feladatokat, a kliens csak a kommunikáció kialakítására szolgál.

6. Tétel

Multiprogramozás és lehetőségei, látszat párhuzamosság, megvalósítások; processzek alapfogalmai (Process Control Blocks, Process Image); processz állapotok (futó, futásra kész, blokkolt, felfüggesztett); processz állapot-átmenet gráfok; processz váltás lépései; Szálak (thread-ek) és használatuk (POSIX rendszerben).

Multiprogramozás és lehetőségei, látszat párhuzamosság, megvalósítások

A **multiprogramozás** lényege, hogy **több processzt** képes látszólag egyidejűleg elvégezni akár egyetlen CPU-val. Ezt úgy oldja meg, hogy egymás után valamilyen ciklusban kerülnek végrehajtásra az egyes processzek. Így nem egy időben hajtódik végre a processzek végrehajtása, de úgy tűnik és így megvalósul. Ha több processzorunk van, akkor a valós multiprogramozást is el tudjuk érni.

Drikert program	**Állapotgépest technika**	**BgProc-os technika**	**Task Switching**
sok feladat kiszolgálására, minden task egy kis feladatot lát el.	bonyolultabb task-ok esetén. Különböző állapotok követhetik egymást egy megadott dolog hatására, mondjuk egy gomb benyomása-	bármelyik hosszú ideig tartó folyamat időről-időre hívja meg a BgProc-ot a melyben a háttérfolyamatok állapotgépek segítségével adminisztrálásra kerülnek.	Timer Interrupt-on időről időre kicseréljük a CPU teljes állapotát, ennél a hardware ismerete nagyon fontos.

processzek alapfogalmai (Process Control Blocks, Process Image)

Processz: Egy program a számítógép memóriájába töltött, futtatás alatt álló példánya. A modern, többfeladatos operációs rendszerek több folyamat párhuzamos futtatására is képesek.

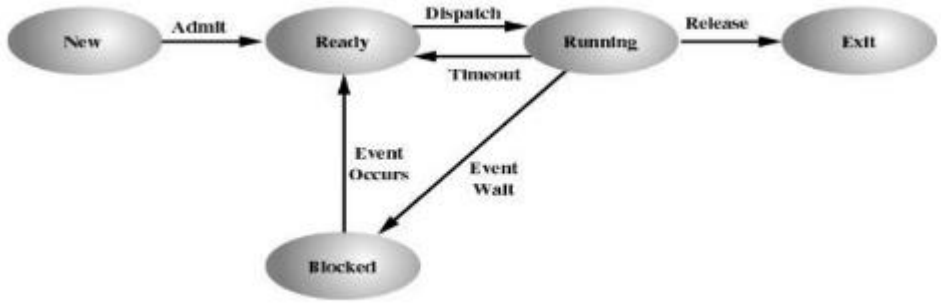
Process Control Blocks :Processz felismerést végzi: Ezt azonosítókkal végzi el, ilyen azonosító a pid ami a kurrens process azonosítója, a ppid a szülő processzt azonosítja az uid a felhasználót a gid pedig a csoportot azonosítja. Ez fontos a multiprogramozásnál, hiszen ez által tudja felismerni és megkülönböztetni a CPU az egyes processzeket.

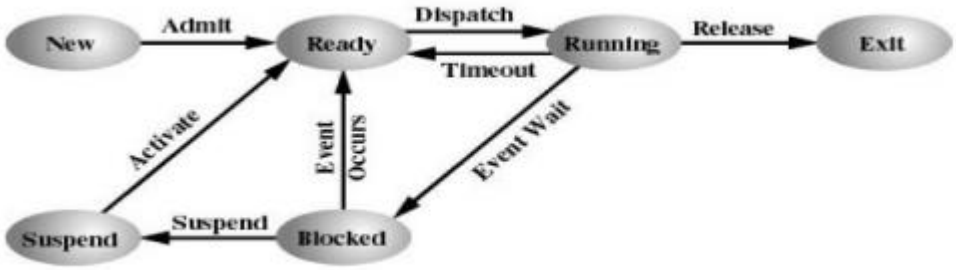
Process Image :virtuális memóriában lévő programkód különböző adatai tartalmazza, ami legalább 4 szokott lenni, a memóriába betöltés előtt szedi össze a rendszer és tárolja el az image-be.

processz állapotok (futó, futásra kész, blokkolt, felfüggesztett)

Futó állapot	Futásra kész állapot	Blokkolt állapot	Felfüggesztett állapot
A processzé a CPU és végzi a feladatát	A process számára minden erőforrás rendelkezésre áll kivéve a CPU-t	A process a CPU-n kivül valamilyen erőforrást igényel még.	Ga sokáig blokkolt akkor kikerülhet a diksre, de ha sokáig van ready állapotban akkoris történhet ugyan ez.(memória szűkösség, időzítés vagy szülő folyamat kérés)

processz állapot-átmenet gráfok





processz váltás lépései

1. **Mentés: PC és a kontextus**
2. **A jelenlegi PCB frissítése**
3. **PCB állapot ready-re váltása**
4. **Ütemező kiválaszt egy másik processzt**
5. **Az új processz PCB állapotának frissítése running-ra**
6. **memória manager adat struktúrájának frissítése**
7. **kontextus és PC visszaállítása.**

Szálak (thread-ek) és használatuk (POSIX rendszerben).

A szál egy **ultra könnyű processz**. A **CPU használat alapegysége**, egy szekvenciálisan futó instrukciósorozat, van dinamikus kontextusa. A vezérlés mindig **a nem blokkolt állapotú thread**-ek között kerül kiosztásra addig ameddig a processz időszeletében arra lehetőség van. Minden szál különböző feladatot lát el, különböző eseményre végezve el azokat.

7. Tétel

Processzek közti kommunikáció (IPC); Kommunikációs alapfogalmak: blokkolás, szinkron, címzés, számosság, szimmetria, közvetettség, puffer kapacitás; primitív mechanizmusok (csatorna, medence); példák (signálok, környezeti változók, fájlokon keresztüli kommunikáció, stb.); osztott memória; üzenetsor.

Processzek közti kommunikáció (IPC)

IPC=Inter Process Communication. Független processz modell a valóságban nem létezik, mindig van a processzek között kapcsolat, legalább az erőforrásokért lévő versenyekért.Szerepe az információmegosztás, munka megosztás és hozzáférések elkülönítése. **IPC** kommunikációhoz 2 rendszerhiváskel **send** és **receive**.

Kommunikációs alapfogalmak: blokkolás, szinkron, címzés, számosság, szimmetria, közvetettség, puffer kapacitás

Blokkolás: Két processz futásának relatív sebességétől függően előfordulhat, hogy a második processznek várnia kell, amíg az első az output-ját elkészíti.A második blokkolt, amíg az inputja elkészül.

Szinkron: Ha mind a küldés, mind a fogadás blokkolásos, akkor a kommunikáció szinkron. Ha valamelyik (vagy mindkettő) nem blokkolásos, akkoraszinkron.

Címzés: Címzésnél kifejezett vagy ki nem fejezett nevek szerepelnek. A címzés segítségével tud a küldő eljuttatni adatokat a fogadónak, a fogadó címe alapján tudja azt kijelölni és felvenni vele a kapcsolatot.

Számosság: A kommunikációban résztvevők száma. A kifejezett neveket használó kommunikáció két processz között.

Szimmetria: Megadja a kommunikáció irányát. Lehet szimmetrikus, amikor több irányú a kommunikáció és lehet aszimmetrikus amikor egy irányú a kommunikáció és általában ez szokott lenni.

Közvetettség: Van közvetlen és közvetett átvitel. A közvetlen átvitelnél a processzeknek ismerniük kell egymást, egymás címeit, ez leginkább két processz között szokásos. A közvetettnél viszont létezik egy **közvetítő entitás**, mint például az **üzenetsor**, amit a kommunikációban résztvevőknek ismerniük kell**.**

Puffer kapacitás: Egy kommunikációs kapcsolatnak lehet zéró, korlátozott vagy végtelen pufferelési kapacitása. Egy csatorna kapacitása a buffer kapacitása.

primitív mechanizmusok (csatorna, medence)

Csatorna	Medence
Az üzenet eltűnik a csatornából, ha kivesszük, vagyis fogadjuk az üzenetet. Csatorna lehet **közvetlen** vagy **közvetett**, vagy **pufferes** változata lehet **LIFO** vagy **FIFO**. A **LIFO** csatorna lényege, hogy az **utolsónak** beérkezett üzenet fogja **először elhagyni** a csatornát, mint egy veremben. A **FIFO**-nál pedig mint egy csővezetékben az **elsőnek** beérkezett üzenet hagyja el **először a csatornát**.	Ennél a kivett üzenet megmarad a medencében (pool) így többször is kivehető. Szokásosan közvetett jellegű, lehet pufferes változata is és szokásosan szimmetrikus. A működése úgy néz ki, hogy az adat kikerül, a medencéből elvégzi a feladatát majd ahelyett, hogy törlődne, visszakerül a medencébe, így többször is hozzáférhetővé válik. Ez a megoldás nagy teljesítmény növekedéshez vezethet.

példák (signálok, környezeti változók, fájlokon keresztüli kommunikáció, stb.)

Signálok: Aszinkron, két processz közt aszimmetrikus, közvetlen puffer nélküli csatorna, gyors és kis információ mennyiséget visz át. A **pause** (), **sleep**(), **usleep**() és **alarm**() rendszerhívások akkorhasznosak, ha szignálozással akarunk processzeket szinkronizálni.A **pause**() feladata: blokkolja a hívó processzt, amíg az nem kapvalamilyen szignált. Miután megkapja a szignált, a pause -1-gyel,**errno=EINTR**-rel tér vissza. A **sleep**() rendszerhívás az argumentumában adott másodpercigblokkol. Tulajdonképpen nem a szignálozáshoz tartoznak asleep-ek, de hasznosak a szinkronizációs ütemezésekben. Az **alarm**(**sec**) feladata: a hívó processz számára megadottmásodperc(sec) múlva **SIGLARM** sorszámú szignált generál. Haa sec 0, a megelőzőleg kért alarm kívánság törlődik.

Környezeti változók: Aszinkron, két processz közötti aszimmetrikus, puffer nélküli medence, gyors, kevés információt visz át. Megvalósítás: A burok környezeti változóit a processzek lekérdezhetik **getenv**() rendszerhívás.

Fájlokon keresztüli kommunikáció: Több processz ugyanabba a file-ba ír, file-ból olvas. Megvalósítás: a szokásos I/Orendszerhívásokkal: open() read() write() close()

osztott memória

- Rendszerhívások:
  - shmget(): létrehozás, azonosítás
    - int shmget(key_t key, int size, int msgflg) ahol a key a külső azonosítóa size a szegmens mérete az msgflg pedig a védelem, hozzáférés beállítás
    - A visszaadott érték az osztott memória belső azonosítója egy változó vagy hibaérték
  - shmat(): processz címtartományára csatolása
    - void *shmat(int id, const void *addr, int flg) ahol az id az azonosító, az addr a leképzési cím az flg pedig a védelem átállás
    - A visszatérési érték siker esetén a szegmens leképzett címe, hiba esetén -1. A visszaadott címet adott típusú mutató veheti fel ezzel a szegmens rekeszeit az adott típusú adatként láthatjuk.
  - shmdt(): lecsatolás
    - int shmdt(const void *addr) ahold addr a rákapcsolt szegmens címe
    - viszatérési érték siker esetén 0, hiba esetén -1
  - shmctl(): kontroll, jellemzők lekérdezése, beállítása, megszüntetése
    - int shmctl(int id, int cmd, struct shmid_ds *buf) ahol az id az azonosító a cmd az állapot lekérdező/beállító a buf pedig az állapotleíró mutató
    - visszatérési érték siker esetén 0, hiba esetén -1

üzenetsor.

Az üzenetsorok aszinkron kommunikációs protokollt valósítanak meg, ami aztjelenti, hogy a küldőnek és a fogadónak nem szükséges egyszerre interakcióban lennie az üzenetsorral. Az üzenetek addig tárolódnak a sorban, amíg a fogadó nem fogadja azokat. Az üzenetsoroknak implicit vagy explicit limitjük van arra az adatmennyiségre, amely egy üzenetben továbbítható, illetve az üzenetek számára, amelyek a sorba elhelyezhetők. Gyors közepes átvihető információ mennyiséget tesz lehetővé. Indirekt FIFO csatorna, többirányú és elvileg végtelen puffer kapacitású, változó üzenethosszúságot is támogatja.

Rendszerhívások:

- msgget(): létrehozás, azonosítás
  - int msgget (key_t, int msgflg), key a kulcs a külső azonosításhoz az msgflg pedig védelem, hozzáférés beállítás.
  - A visszaadott érték az üzenetsor belső azonosítója vagy egy hibaérték lehet.
- msgctl(): kontroll, jellemzők lekérdezése, megszüntetés
  - int msgctl(int id, int cmd, struct msqid_ds *buf), az id a sor belső azonosítója, a cmd egy állapot lekérdező/változtató, a buf pedig a felhasználói címtartományhoz tartozó állapotleíró struktúra mutatója
  - Siker esetén a visszatérési érték 0, hiba esetén -1
- msgnd(): üzenet betétel a sorba
  - int msgnd(int id, struct msgbuf *msgp, size_t size, int msgflg), id a belső azonosító, msgp az üzenetstruktúra mutató, size az üzenet hossz az msgflg pedig egy flag, a size pedig az üzenet hossza
- msgrcv(): üzenet kivétel sorból
  - int msgrcv(int id, struct msgbuf *msgp, size_t size, long msgtyp, int msgflg), id az azonosító, msgp az üzenetstruktúra mutató, size az üzenet hossza, msgtyp a típus a szelektáláshoz, ami lehet =<>0

8. Tétel

Időkiosztás (scheduling), ütemezési stratégiák; CPU ütemezés eszközei; Ütemezési döntési helyzetek; Prioritást befolyásoló tényezők; Processzek életszakaszai; Ütemezési algoritmusok: FCFS, SJF, Policy Driven Scheduling, RR, Multilevel Feedback Queue Scheduling; öregedési algoritmus; példák (Vax, Unix, Linux, NT).

Időkiosztás (scheduling), ütemezési stratégiák

Időkiosztásra azért van szükség, mert több processz van, mint amennyi processzor így a processzek osztoznak a processzor idején. Ez egy olyan eszköz,amiben a processzek hozzáférést nyernek az erőforrásokhoz egy megadott időre.

Long Term Sceduling:	Medium Term Scheduling:	Short Term Scheduling:
JOB ütemezés: JOB pool-ból melyiket választjuk ki futásra, befolyásolja a multiprogramozási szintet. Ez álítja, hogy melyik processz futhat a rendszerben. Ritkán kell futnia, egy processz befejeződésekor választ ki egy új elindítandót	suspended processzek közül választja ki, hogy melyeket tegye futásra késszé azaz Ready-vé.időszakos terhelés ingadozást próbálja kompenzálni a felfüggesztésekkel Alacsony prioritású processzeket is felfüggeszhet	Futásra kész processzek közül választja ki, hogy melyik kapja meg a CPU-t gyors működés Megszakításokra vagy rendszerhívásra reagál

CPU ütemezés eszközei

CPU több eszközzel rendelkezik, amivel képes ütemezni a feladatok végrehajtását, az egyik ilyen fontos eszköz az óra (**clock**). Ez periodikusan megszakításokat generál és az időt szeletekre bonthatjuk vele, ezzel pedig számon tarthatjuk a processzek életidejét, felhasználási idejét.(oszcillátor kristály, számlálótartó regszter). A másik fontos eszköz az Időzítő: A **watchdog-Timer** túlcsordulást figyeli, mindig lenullázódik az értéke, ha nem akkor baj van és jelez.

Ütemezési döntési helyzetek

3 féle döntési stratégiát különböztethetünk meg:

1. **nem beavatkozó (non preemptive)**
2. **szelektív beavatkozó**
3. **szelektív beavatkozó(preemptive)**

Négyféle döntési helyzet lehet:

1. Amikor egy process futó állapotból blokkolt állapotba megy
2. Amikor egy processz futó állapotból futásra kész állapotba megy
3. Amikor egy processz blokkolt állapotból futásra kész állapotba megy
4. Egy process törlésekor

Prioritást befolyásoló tényezők

Prioritás segítségével lehet különböző ütemezéseket beállítani a processzek

fontosságát figyelembe véve. Ezt több dolog is befolyásolhatja:

- processzek memóriaigénye
- processz érkezési ideje vagy sorrendje
- Eddigi CPU felhasználási idő: CPU lázas processzek az I/O lázas prcosszekkel szemben
- Várható CPU felhasználási idő: számolás igényes processzek szemben I/Oigényes processzek
- processz időszerűsége: CPU lázas szakaszok szemben I/O lázas szakaszok

Processzek életszakaszai

Ütemezési algoritmusok: FCFS, SJF, Policy Driven Scheduling, RR, Multilevel Feedback Queue Scheduling

FCFS: Az egyik algoritmus az igény bejelentés alapján alakítja a sorrendet, ez a FCFS vagyis a First Come First Served.

SJF: Shortest Job First algoritmus, mely a legkissebb igényűt veszi előre. Régi algoritmus, de egyszerű. A nyomtatók ütemezésére még ma is alkalmazzák.

Policy Driven Scheduling: Valamilyen ígéret szerint ütemez. Egy reális ígéret, hogy n számú process esetén 1/n-ed részét kapja a processz a CPU-nak.

RR: Round Robin körkörös sorrendet alakít ki. Egyszerű és elég régi algoritmus.Processzekhez időszeleteket rendelünk.

Multilevel Feedback Queue Scheduling: Több összetevőtől függ a sorrend kialakítása. Több ready sor van különbözőprioritási szintekhez. Elvileg mehet minden sorhoz más scheduling algoritmus.

öregedési algoritmus

“Aging “ algoritmus

0, 1, 2, ...,i,.. CPU lázas szakaszok ismétlődnek (nem egyforma időt igényelve). Becsülhetjük a következő idejét (T_i : tényleges, mért; B_i : becsült):

$B_{i+1} = a * T_i + (1 - a) * B_i$; ahol $0 <= a <= 1$.

Legyen $a = 1/2$, ekkor (a: súlyozó faktor)

$B_{i+1} = (T_i + B_i) / 2$

		i	i +1	i + 2	i + 3	i + 4	i + 5
T_i	6	4	6	4	13	13	
B_i	10	8	6	6	5	9	11

példák (Vax, Unix, Linux, NT).

VAX:

- Multilevel jelegű mely alkalmas Real Time-re is
- 32 prioritási szintje van, minél nagyobb a szám annál nagyobb a prioritás
- 16-32 a valós idejű processzek számára van fenntartva
- 0-15-ig pedig a reguláris processzeknek

UNIX:

- A processzeknek User vagy Kernel mode prioritás értéke lehet. Az előbbi dinamikus, utóbbi fix érték
- Küszöb prioritási szint
- A user_mode prioritások dinamikusan igazodnak
- A p_cpu mező a futó processznél a CPU használattal arányosan nő.

Linux:

- Több ütemezési osztály van: SCHED_FIFO, SCHED_RR,SCHED_OTHER
- Schedule függvény hívódik, ha a processz blokkolt állapotba megy és minden syscall és megszakítás után, ha egy flag be van állítva.
- schedule függvény 3 dolgot csinál: a szokásos ütemezési teendőket, alegmagasabb prioritású processz meghatározása, Context Switch

NT:

- Van standby állapota: a legnagyobb prioritású futásra kész
- 32 prioritási szint: 16-31 valós idejű, 0-15 normál mint a VAX-nál
- A lejárt quantum szál prioritása 1-el csökken a bázisig
- A kész állapotba jutó prioritása növekszik a blokkolódás típusától függetlenül

9. Tétel

Versenyhelyzetek: Konkurens folyamatok közötti kommunikáció, versenyhelyzet, kritikus szekció fogalma, kritikus szekció sikeres megvalósításának a feltételei. Konkurrens programozás alapjai: megszakítások tiltása, zárolásváltozó, szigorú alternáció, Peterson módszere, Test and SetLock utasítás; prioritás inverzió; gyártó- fogyasztó probléma; szemaforok, mutexek, monitorok; 5 filozófus problémája, író- olvasó probléma, alvó-borbély problémája.

Versenyhelyzetek: Konkurens folyamatok közötti kommunikáció, versenyhelyzet, kritikus szekció fogalma, kritikus szekció sikeres megvalósításának a feltételei. Konkurrens programozás alapjai: megszakítások tiltása, zárolásváltozó, szigorú alternáció, Peterson módszere, Test and SetLock utasítás

Versenyhelyzet olyan helyen alakulhat ki ahol az együtt dolgozó processzek közös tárolóterületen osztoznak, amelyből mindegyik olvashat, és amelybe írhat.

Kritikus szekció: A programnak azt a részét, amelyben a megosztott memóriát használja,kritikus területnek vagy kritikus szekciónak nevezzük.

Konkurrens programozás alapjai:

1. **Megszakítások tiltása:** A legegyszerűbb megoldás, hogy minden processz letiltja az összesmegszakítást, mihelyt belép saját kritikus szekciójába és újra engedélyezi, éppen mielőtt elhagyná azt.
2. **Zárolásváltozó:** Ez egy szoftveres megoldás. Tekintsünk egy egyszerű megosztott változót mely kezdetben 0 értékű. Mielőtt a processz belépne, a saját kritikus szekciójába először ezt az értéket vizsgálja meg. Ha 0 értékű, akkor a processz 1-re vált és belép. Ha már 1 akkor addig vár míg a processz 0 nem lesz.
3. **Szigorú alternáció:** Egy ciklus segíti az ütemezést. Egy 0 kezdőértékű turn változó követi nyomon, hogy ki lesz a következő, aki a kritikus szekcióba lép ésvizsgálja vagy aktualizálja a megosztott memóriát. Kezdetben a 0. processz vizsgálja meg a turn értékét, ha 0-nak látja akkor belép a szekcióba.
4. **Peterson módszere:** Két C-ben megírt eljárásból áll, ami azt jelenti, hogy minden definiált és használt függvényhez függvényprototípusokat kell biztosítani. Mielőtt belépne a szekcióba minden processz meghívja az enter_region függvényt, paraméterként átadva a saját processz sorszámát 0-t vagy 1- et. Ez a hívás azt eredményezi, hogyha szükséges, akkor a biztonságos belépésig várakozni fog. Miután végzett a szekcióban a processz meghívja a leave_region-t jelezve, hogy végzett és megengedi a másik processznek, hogy belépjen. Bővebben a 0. process meghívja az enter_regiont amely jelzi az érdekeltségét majd a turn változót 0-ra állítja. Ha az 1. processz nem érdekelt akkor az enter_region azonnal visszatér. Ha az 1. processz meghívja az enter_region-t addig vár míg az interested[0] FALSE-ra vált, ami csak akkor következik be ha a 0. processz meghívja a leave_regiont
5. **Test and Set Lock utasítás:** A LOCK memóriaszó tartalmát az RX regiszterbe, és ezután egy nem 0 érétket ír erre a memória címre. A szó kiolvasása és a tárolási művelet garantáltan nem választható szép így az utasítás végéig más processzor nem érheti el a memóriaszót. Ekkor zárolja a sínt és nem engedi meg hogy más processzorok hozzáférjenek.

prioritás inverzió

A Peterson és TSL megoldások arra épülnek, hogy figyeljenek egy változót, ami megállapítja, hogy beléphetnek-e, de ezek sok CPU időt elveszenk és váratlan hatásokat is generálhatnak, az egyik ilyen hatás a Fordított Prioritás.

gyártó- fogyasztó probléma

Két processz osztozik egy közös, rögzített méretű tárolón. Az egyikük a gyártó, amely adatokat helyez el benne, a másik a fogyasztó, amely kiveszi azokat. A probléma akkor jelentkezik, amikor a gyártó új elemet akar elhelyezni a tárolóba, de az már tele van. Erre a megoldás, hogy a gyártó alvó állapotba kerül, míg a fogyasztó ki nem vesz 1-2 elemet a tárolóból ez fordítva is igaz.

szemaforok, mutexek, monitorok

szemafonok: A szemafor a számítógép-programozásban használt változó vagy absztrakt adattípus, amit az osztott erőforrásokhoz való hozzáférések szabályozásához használnak a többszálú környezetekben.

mutexek: Amikor a szemafor számlálási képességeire nincs szükség, akkor a szemafor egy egyszerűsített változata, a mutex kerülhet felhasználásra.

monitorok: Magas szintű szinkronizációs primitív. A monitor eljárások, változók és adatszerkezetek együttese és mindezek egy speciális fajta modulba vagy csomagba vannak összegyűjtve.

5 filozófus problémája, író- olvasó probléma, alvó-borbély problémája.

5 Filouófus: Öt filozófus ül egy kerek asztalnál, mindegyik filozofusnak van egy spagettije.A spagetti olyan csúszós, hogy minden filozófusnak 2 villára van szükség.Minden egymás melletti tányér között van egy villa. A filozófus valamilyensorrendben megpróbálja megszerezni a bal és jobb oldalán lévő villát, ha sikerültmegszerezni, akkor eszik, egy ideig majd leteszi, a kérdés, hogy lehet-e olyanprogramot csinálni, ami az elvárások szerint működik, és sose akad el. A **take-fork eljárás megvárja**, amíg a megadott villa szabad lesz, és ekkor megszerzi, detegyük fel, hogy egyszerre mind az 5 felveszi a bal oldalit és holtpont alakul ki,mivel egyik se tudja a jobb oldalit felvenni és ezzel kialakul a probléma. Egy lehetséges megoldás, ha véletlen ideig és nem ugyanannyi ideig vár a jobb oldalivilla megszerzésére, ugyanis akkor kicsi lenne az esély, hogy ugyanabban azütemben folytatódjanak az események 1 órán keresztül. Ha nem sikerülmegszerezni, akkor nem egyből próbálja megint megszerezni, hanem egyvéletlen ideig várakozik.

író- olvasó probléma: Például sok versengő processz akik szeretnék olvasni vagy írni az adatbázist.Elfogadható, hogy több processz egyidőben olvassa az adatbázist, de ha egyprocessz írja az adatbázist, akkor azt más processzeknek nem szabad elérniük, mégaz olvasóknak sem. A kérdés, hogyan programozzuk le az olvasókat és az írókat.Egy lehetséges megoldás, hogy az első olvasó amelyik belép végrehajt egy down-ta szemaforon, a következő olvasók pedig a számlálót növelik. Ha egy olvasó kilépakkor csökkenti a számlálót és az utolsó kilépő végrehajt egy up-ot a szemaforon,lehetővé téve egy blokkolt írónak, hogy belépjen. Ennek egy továbbfejlesztettváltozata az, hogy amikor egy olvasó érkezik és egy író már vár az olvasófelfüggesztődik az író mögött, ahelyett hogy rögtön bekerülne. Így az írónak csakazt kell megvárni, hogy az előtte lévő olvasók végezenek, de az utána lévő olvasókatmár nem, ezzel kijavítva az előző megoldás hibáját amiben előfordulhat, hogy az írósose kerül sorra.

alvó-borbély problémája: TODO

10. Tétel

Holtpont definíciója; Holtpont kialakulásának feltételei (Coffman); Példa holtpont kialakulására; Holtpont feloldási stratégiák (struccpolitika, felismerés és helyreállítás, megelőzés, dinamikus megoldás); példák; Bankár algoritmus; Dijkstra algoritmus.

Holtpont definíciója

Egy csoport processz holtpontban van, akkor ha a csoport minden processze egy olyan erőforrás birtoklására vár, amelyet egy másik a csoporton belüli process birtokol.

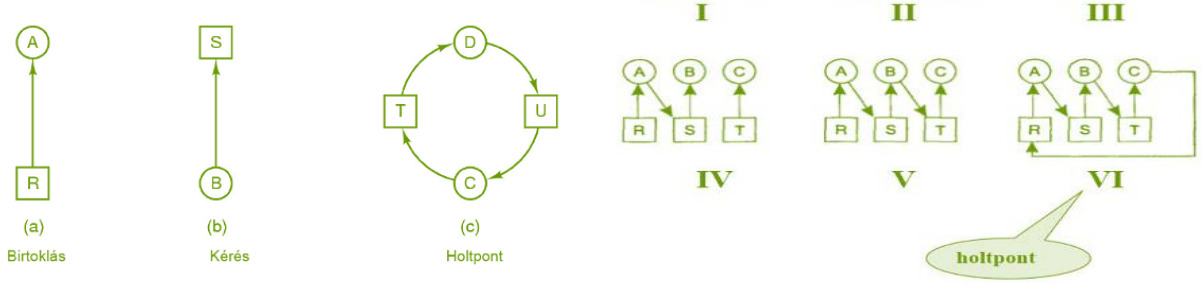
Holtpont kialakulásának feltételei (Coffman)

Coffman szerint 5 feltétel szerint alakulhat ki holtpont:

1. Mutex: minden erőforrás vagy hozzá van rendelve (locked) egy processzhez vagy szabad
2. Birtoklás: Egy processz birtokolhat, erőforrást majd birtokolhat újabbakat
3. Megszakíthatatlanság: egy processz nem veszítheti el az általa birtokolt erőforrást
4. Ciklikusság: kör-kör várakozások alakuljanak ki az erőforrásokon.
5. Ha ezen feltételek valamelyike nem teljesül akkor nincs holtpont helyzet.

Példa holtpont kialakulására

- Erőforrások: A,B,C,D
- Processzek: R,S,T,U



Holtpont feloldási stratégiák (struccpolitika, felismerés és helyreállítás, megelőzés, dinamikus megoldás)

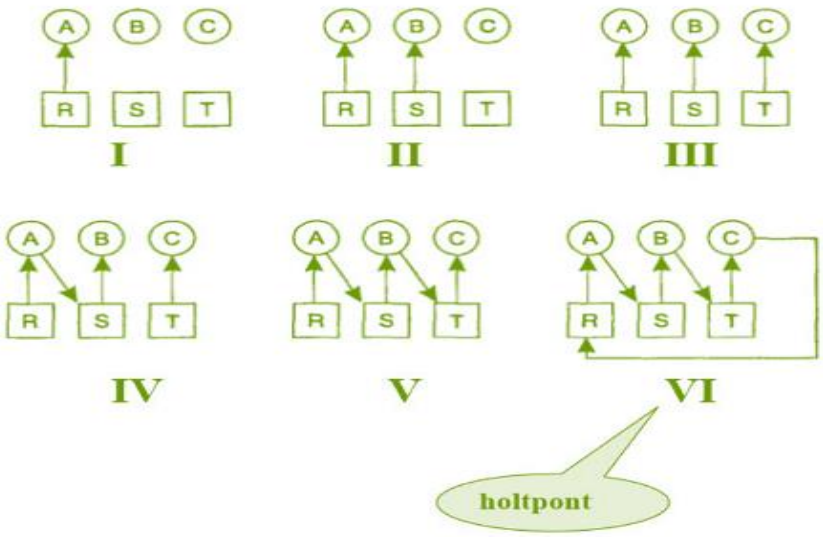
Struccpolitika: Ennél a stratégiánál nem foglalkozunk a holtponttal és valószerűtlen eseménynek tekintjük a helyzetet. Unix-nál alkalmazzák.

Felismerés és helyreállítás: Ebben a stratégiában, ha kör alakulna ki egy kéréskor, akkor megszüntetjük a processzt. Mégpedig úgy, hogy azokat a processzeket amelyek hosszú ideig allokálnak erőforrást megszüntetjük.

Megelőzés: Egy másik megoldás, hogy a processzek egy előre megadott sorrendben igényelhetik az erőforrást, de ennek hátránya, hogy jó sorrendet nem igazán lehet kialakítani. Példa sorrendre: 1. erőforrás nyomtató, 2. erőforrás szalag, 3. erőforrás lemez,4. erőforrás robot

Dinamikus megoldás: Egyetlen erőforrásra koncentrál. Lényege, ha jelen pillanatban ki tudjuk elégíteni valamely processz maximális igényét, akkor biztonságos állapotban vagyunk, ha viszont nem akkor bizonytalan az állapot.

példák



Bankár algoritmus

Egy erőforrásnál alkalmazzák. Az algoritmus csak akkor engedélyez átmenetet ha biztonságos állapothoz vezet. Az algoritmus feltételezi, hogy minden folyamat az indulásakor előre be tudja jelenteni az operációs rendszernek, hogy melyik erőforrásból legfeljebb mennyit fog a működése során használni.

Hitel igenylo	Birtokol	Max
A	0	6
B	0	5
C	0	4
D	0	7

szabad: 10
biztonságos állapot

Hitel igenylo	Birtokol	Max
A	1	6
B	1	5
C	2	4
D	4	7

szabad: 2
biztonságos állapot

Hitel igenylo	Birtokol	Max
A	1	6
B	2	5
C	2	4
D	4	7

szabad: 1
bizonytalan állapot

Dijkstra algoritmus.

A Dijkstra-algoritmus egy mohó algoritmus, amivel irányított vagy irányítás nélküli gráfokban lehet megkeresni a legrövidebb utakat egy adott csúcspontból kiindulva. Az algoritmust Edsger Wybe Dijkstra holland informatikus fejlesztette ki