Operációs rendszerek be és kivitelkezelése, holtpont fogalma, kialakulásának feltételei, holtpontkezelési stratégiák, bankár algoritmus.

Input/Output

I/O Hardware

I/O eszközök (kommunikációs portok szerint is lehet csoportosítani):
- Blokkos eszközök
- Karakteres eszközök

Eszközvezérlők (I/O eszközök elektromos komponensét alkotják):
- IDE: Integrated Drive Electronics
- SCSI: Small Computer System Interface

IRQ: Interrupt Request
DMA: Direct Memory Access

I/O Software

Réteges szervezés

Felhasználói szint
Eszközfüggetlen Operációs rendszer
Eszközmeghajtók
Megszakításkezelés

Holtpont

Az erőforrás kezelésnek a köv. módszere végtelenül egyszerűnek tűnik: ha van szabad erőforrás kielégítjük, ha nincs, a folyamat várakozni kényszerül. Ez a probléma nagyvonalú megközelítése, azonban korlátai elég hamar megmutatkoznak.
Veszélye: Több folyamat egy olyan erőforrás felszabadulására vár, amit csak egy ugyancsak várakozó folyamat tudna előidézni. Ez az állapot kapta a HOLTPONT (deadlock) nevet. Holtpont esetén a folyamatok körkörösen egymásra várakoznak.
Ha megszüntetjük a párhuzamosságot a probléma megelőzhető.

A holtpont kialakulásának feltételei

Kölcsönös kizárás Vannak olyan erőforrások a rendszerben, melyeket a folyamatok csak kizárólagosan használhatnak.

Foglalva várakozás legyen olyan folyamat mely lefoglalva tart erőforrásokat, miközben más erőforrásokra várakozik.

Nincs erőszakos erőforrás-elvétel a folyamatok addig birtokolják az erőforrást, míg saját jószántukból fel nem szabadítják azokat.

Körkörös várakozás Létezik a rendszerben egy olyan folyamatsorozat, melyben minden folyamat az utána következő folyamat által foglalt erőforrásra vár, a sorozat utolsó tagja pedig a sorozat első tagjára.

Modell: Irányított gráf

Processzus

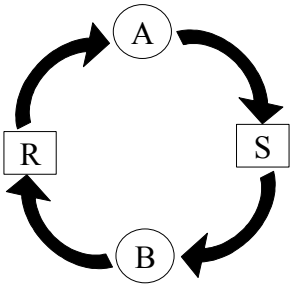
A Processzus használja R erőforrást

Erőforrás

Processzus

B Processzus igényli S erőforrást

Erőforrás



Körkörös várakozás

Holtpontkezelési stratégiák

Nem veszünk tudomást a holtpontról (**strucc algoritmus**) Ha a holtpont kialakulásának valószínűsége kicsi, a rendszer újraindításának nincsenek kritikus következményei, akkor megérheti ezt a megoldást választani.

Észrevesszük, ha a holtpont kialakult és megpróbáljuk feloldani

Vagy az erőforrást vesszük el a folyamatoktól, vagy kilőjük a folyamatokat. Mindkettő nagy kárt okozhat, mert a kilőtt folyamatokhoz tartozhatnak megnyitott fájlok, stb.

Védekezünk a holtpont kialakulása ellen:

Megelőzzük a holtpont kialakulását: olyan rendszert tervezünk, ahol nem teljesülnek a holtpont feltételei, így elvileg sem lehet holtpont. Ahhoz, hogy a rendszerben holtpont kialakulhasson, négy feltétel egyidejű teljesülése szükséges:

Kölcsönös kizárás minimálisra csökkentése: lehetőleg többpéldányos erőforrásokat alkalmazunk, ahol ez nem lehetséges, ott a hozzáférést megpróbáljuk oszthatatlan műveletté tenni.

Foglalva várakoztatás megszüntetése: Ha minden folyamat betartja a szabályt, miszerint az egyidejűleg szükséges valamennyi erőforrását egyetlen rendszerhívással kéri el, akkor elkerülhető a foglalva várakoztatás. Ennek ára van: az erőforrás-kihasználtság romlása.

Nincs erőszakos erőforrás-elvétel kiküszöbölése: Ha menthető állapotú erőforrásaink vannak, akkor megtehetjük, hogy elvesszük egy adott folyamat erőforrását és egy másiknak adjuk, majd annak lefutása után visszaadjuk a régi állapotában az erőforrást az első folyamatnak.

Körkörös várakozás megakadályozása: A folyamatok megegyeznek az erőforrások sorszámozásában, minden folyamat csak nagyobb sorszámú erőforrást igényelhet azoknál az erőforrásoknál melyeket birtokol. Ekkor biztosan nem alakulhat ki kör.

Ha a négy feltétel legalább egyikének a kialakulását megakadályozzuk, nem alakul ki holtpont! Az első és a harmadik feltétellel nem tudunk mit kezdeni, hiszen a rendszerünkben vagy vannak kölcsönös kizárást igénylő ill. erőszakkal el nem vehető erőforrások, vagy nincsenek. Tehát csak a várakozás közbeni foglalásra és a körkörös várakozásra érdemes koncentrálni.

Az **egyetlen foglalási lehetőség** stratégia a várakozás közbeni erőforrás lekötést tiltja meg. A folyamat csak egy lépésben (célszerű induláskor) foglalhatja le az összes szükséges erőforrást. Ha bármelyik erőforrás is hiányzik a folyamat várakozó listára kerül. Az a folyamat, amelyik már rendelkezik erőforrással, többletigényt nem nyújthat be, a további kéréseket az operációs rendszer elutasítja.

Holtpont nem alakulhat ki, mivel a futó folyamatok nem kényszerülnek várakozni, mindenük megvan, a várakozó folyamatok pedig nem rendelkeznek erőforrásokkal..

Előnye: egy folyamat, ha már megszerezte az összes erőforrást, soha többé nem kell, hogy erőforrásra várakozzon, azaz gyorsan lefuthat.

Hátrányai: a folyamatok olyankor is kénytelenek foglalva tartani erőforrásokat, ha pillanatnyilag nincs rá szükségük. Nem mindig mérhető fel előre egy folyamat erőforrás igénye. Ha egy folyamat sok és népszerű erőforrást igényel, előfordulhat, hogy soha nem jön el az a pillanat, amikor mindegyik egyidejűleg áll rendelkezésre, így a folyamat bizonytalan ideig várakozni kényszerül, *éhezik.*

A **rangsor szerinti foglalás** a ciklikus várakozás lehetőségének kiküszöbölésével alkalmas a holtponthelyzetek megelőzésére. Lényege, hogy az erőforrások osztályaihoz egy-egy növekvő sorszámot rendelünk úgy, hogy a leggyakrabban használt erőforrások kapják a legkisebb sorszámot. Ha egy folyamatnak egy osztályon belül több erőforrásra van szüksége, azokat csak egyszerre igényelheti. Egy folyamat csak olyan osztályból igényelhet erőforrást, melynek sorszáma magasabb, mint a már birtokolt erőforrások sorszáma. Ha egy folyamatnak a meglévőknél alacsonyabb sorszámú erőforrásra van szüksége, fel kell szabadítania erőforrásokat egészen addig a szintig, míg az igénylési feltételek nem teljesülnek.

A **bankár algoritmus** a rendszert mindig ún. biztonságos állapotban tartja. Egy állapotot akkor tekintünk *biztonságosnak*, *ha létezik legalább egy olyan sorozat, amely szerint az összes folyamat erőforrás igénye kielégíthető*. Sose elégítsünk ki egy igényt, ha az nem biztonságos állapotot eredményez.

Az algoritmus csak akkor működik, ha a folyamatok indulásukkor tudják, hogy a különböző erőforrásokból egyszerre maximálisan hányat fognak igényelni, és ezt az op. rendszernek – egy rendszerhívás által – be is jelentik. Ez a módszer legnagyobb hátránya, hogy vannak olyan folyamatok, amelyek erőforrás igénye előre nem tudható.

Meg kell jegyezni, hogy ha egy rendszer állapota nem biztonságos, nem jelenti azt, hogy a holtpont kialakul, hanem mindössze azt, hogy *lehet hogy* kialakul! Azaz az algoritmus úgy

garantálja a holtpont mentességet, hogy már egy, a holtpontnál tágabb szituáció a nem biztonságos állapot – kialakulását is megakadályozza.
Bankár: op. rendszer
Ügyfél: processzus
Hitelegység:erőforrás

	Birtok	***Max***
x	0	6
y	0	5
z	0	4
w	0	7

Szabad: 10

Biztonságos állapot

	Birtok	***Max***
x	1	6
y	1	5
z	2	4
w	4	7

Szabad: 2

Nem biztonságos állapot

	Birtok	***Max***
x	1	6
y	2	5
z	2	4
w	4	7

Szabad: 1

Holtpont felszámolása

Sajnos a holtpontmegelőző stratégiák mindegyikének komoly elvi és/vagy gyakorlati problémái vannak, tehát ezek csak speciális körülmények között alkalmazhatóak.

Holtpont detektálása

Először is képesnek kell lennünk észrevenni, *detektálni*, ha egy holtpont kialakult. Ehhez az op. rendszernek folyamatosan nyílván kell tartania az erőforrások szétosztását és a ki nem elégített igényeket. Ezen adatokból kiindulva időnként egy ún. holtpont detektáló algoritmust kell lefuttatnia. Mi az az „időnként” , amikor a holtpont detektáló algoritmust futtatni célszerű:

- Minden olyan esetben futtassuk le ezt, amikor egy erőforrásigényt nem tudunk azonnal kielégíteni. Mivel ilyen eset nagyon gyakran történik, ezért ez a megoldás nagyon lelassítaná

rendszerünk működését.
- Áldozunk a biztonságból. Futtassuk a holtpont detektáló algoritmusunkat viszonylag ritkán, időnként periodikusan. Ezzel az a baj, hogy ez alatt az idő alatt, több holtpont is kialakulhat és akkor elég régóta blokkolhatja is a folyamataink működését.

Holtpont megszüntetése

Az első gondolatunk a holtpont megszüntetésére az lehet, hogy az összes, a holtpontban lévő folyamatot szüntessük meg. Ez tagadhatatlanul elég egyszerű megoldás, de hatalmas veszteséggel jár. Valamilyen szempont alapján válasszunk ki először egyet a holtpontban résztvevő folyamatok közül, szüntessük meg azt, majd ellenőrizzük, hogy a maradék még mindig holtpontban van-e.
Áldozat kijelölési szempontok:
- melyikkel hány erőforrást nyerek;
- hány további erőforrást igényel;
- mennyi mér elhasznált CPU időt ill. I/O munkát vesztek a megszüntetéssel;
- mennyi idő van még hátra a futásból;
- ismételhető / nem ismételhető folyamat-e;
- a folyamat prioritása / célszerű kis prioritású folyamatot választani;
- megszüntetése hány további folyamatot érint.

Tovább finomítva. Lehet, hogy nincs is szükség a folyamat teljes megszüntetésére, elég ha néhány erőforrást elveszünk tőle. (Sok esetben egy nem elvehető erőforrás elrablása a folyamat működésében helyrehozhatatlan hibát okoz, míg máskor ez nem okoz problémát.)

Akármilyen módszer szerint is járunk el a holtpont megszüntetésénél, figyelembe kell vennünk azt is, hogy ugyanazt a folyamatot nem választhatjuk akárhányszor áldozatul, hiszen ez a folyamat kiéheztetését okozná.

A holtpontkezelés két legnagyobb veszélye:
- egyik az ún. alul szabályozás, vagyis az, ha olyan túlbiztosításra törekszünk, hogy holtpont ugyan nem lesz, de a rendszer lehetőségeit sem tudjuk kihasználni.
- a másik probléma az ún. túlszabályozás, azaz olyan bonyolult algoritmust választunk, hogy maga az algoritmus futtatása lassítja le a rendszert.