Beágyazott információs rendszerek

Beágyazott rendszerek ellenőrzéstechnikája

Előadásvázlat

Majzik István
Budapesti Műszaki és Gazdaságtudományi Egyetem
Méréstechnika és Információs Rendszerek Tanszék
majzik@mit.bme.hu

Tartalom

1. Termékminőség, szolgáltatásbiztonság

Felhasználó: *Szolgáltatás* jellemzői érdeklik
- szolgáltatásminőség:
  - funkcionális jellemzők
  - nem-funkcionális jellemzők: teljesítmény, időbeliség
  - általános értelemben vett megbízhatóság (rendelkezésre állás, javíthatóság,...)
- termékminőség: ezt nyújtja a tervező (előállítási folyamat, ld. ISO 9000)

Szolgáltatásbiztonság (üzembiztonság) (dependability):

Milyen biztonsággal képes ellátni feladatait a rendszer?

Igazoltan bízni lehet-e a szolgáltatásban.
- igazoltan: elemzésen, méréseken alapul.
- bizalom: szolgáltatás az igényeket kielégíti

Szolgáltatásbiztonság jellemzői:

- megbízhatóság: folyamatos szolgáltatásra kész
- rendelkezésre állás: (javítva) használatra kész
- biztonság: nincs káreset/baleset
- bizalmasság: nincs jogosulatlan információ kiszolgáltatás
- integritás: nincs hibás (belső) állapotváltozás
- karbantarthatóság: javítás és módosítás lehetősége

A szolgáltatásbiztonság befolyásoló tényezői:

- Hibajelenség (failure):
  A specifikációnak nem megfelelő szolgáltatás
  - értékbeli / időzítésbeli, katasztrofális / „jóindulatú”
- Hiba (error):
  Hibajelenséghez vezető rendszerállapot
  - lappangó → detektált
- Meghibásodás (fault):
  A hiba feltételezett oka
  - hatás: alvó → aktív
  - fajta: véletlen vagy szándékos, időleges vagy állandósult
  - eredet: fizikai/emberi, belső/külső, tervezési/működési

A meghibásodás hatáslánc:

- Meghibásodás → Hiba → Hibajelenség
  - pl. szoftver: **állandósult tervezési hibák**
    - meghibásodás: progr. hiba: csökkentés helyett növel
    - hiba: vezérlés ráfut, változó értéke hibás lesz
    - hibajelenség: számítás végeredménye rossz
  - pl. hardver: **többnyire időleges (ritkábban állandósult) hibák**
    - meghibásodás: kozmikus sugárzás egy bitet átbillent
    - hiba: hibás memóriacella olvasása
    - hibajelenség: robotkar a falnak ütközik

Eszközök a szolgáltatásbiztonság növelésére

- Hiba megelőzés: Meghibásodás megakadályozása
  - fizikai hibák: jó minőségű alkatrészek, árnyékolás,...
  - szoftver tervezési hibák: ellenőrzések a tervezés során (verifikáció és validáció)
- Hiba megszüntetés:
  - prototípus fázis: dinamikus ellenőrzések (tesztelés, diagnosztika, javítás)
  - működés közben: monitorozás, javítás
- Hibatűrés: Szolgáltatást nyújtani hiba esetén is
  - működés közben: automatikus hibakezelés
- Hiba előrejelzés: Hibák és hatásuk becslése
  - mérés és „jóslás”, megelőző karbantartás

Az ellenőrzések típusai:

- Verifikáció (igazolás): „Jól tervezem-e a rendszert”
  A tervezett/megvalósított rendszer(modell) megfelel-e a specifikációnak
  - formális verifikáció: a rendszermodell és a követelmények matematikai objektumok
  - statikus analízis, (manuális) átvizsgálás
  - tesztelés
  - szimuláció: rendszer és környezet modellje alapján
- Validáció (érvényesítés): „Jó rendszert terveztem-e”
  A rendszer megfelel-e az elvárásoknak
  - validációs tesztelés: prototípus alapján
  - szimuláció valós környezetben, mérések

2. A fejlesztési folyamatban megvalósuló ellenőrzések áttekintése

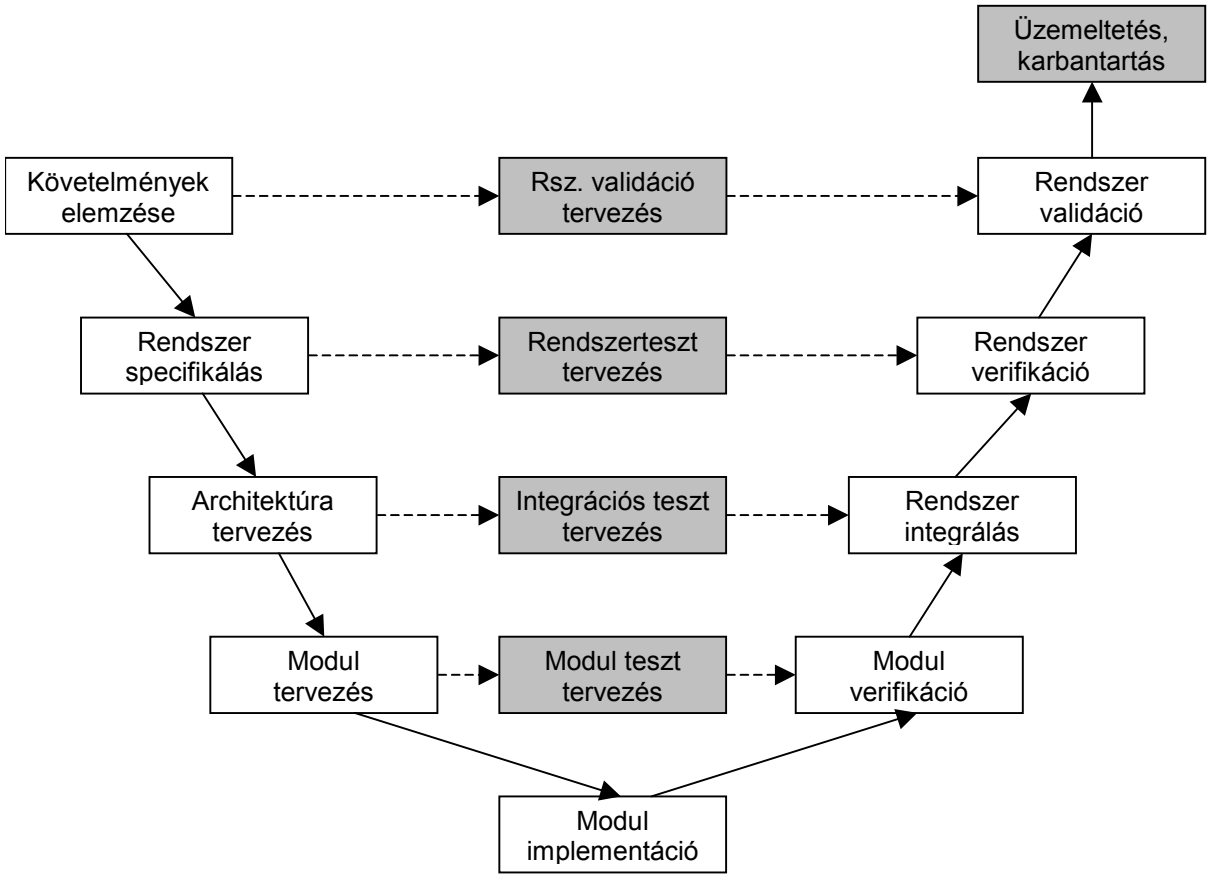
A fejlesztési folyamat jellemzői beágyazott rendszerek esetén:

- jól meghatározott specifikáció, ismert környezet
- költséges az üzembe helyezés utáni javítás (pl. visszahívás kell, nincs „rendszergazda”)
- hibák kockázata nagy, felelősséget kell vállalni (baleset, káreset; jótállás)
- gyakran hatósági engedély is kell az üzembe helyezéshez.

Következmény:

- "Hagyományos", előrelépő, jól definiált fázisokat tartalmazó fejlesztési folyamat (vízesés illetve V-modell a spirál, iteratív, 4G, extrém programozás helyett)
- Szigorú feltételekhez kötött előrelépés az egyes fázisokban: hangsúlyos ellenőrzéstechnika.

A fejlesztési folyamat egyszerűsített V-modellje:



Ellenőrzések a specifikálás és a tervezés során:

- Statikus ellenőrzés (teljesség és ellentmondás-mentesség)
- Formális verifikáció (formális modellek esetén)
- Veszély- és biztonsági analízis

Ellenőrzések az implementációs és integrációs fázisban:

- Tesztelés (dinamikus tesztelés)
- Monitorozás (futásidőbeli megfigyelés)

Ellenőrzések a rendszerátadás előtt:

- Validációs tesztelés

A tipikus ellenőrzési technikák áttekintése:

Fázis	**Szerep**	**V&V szempontja**	**V&V eszköze**
Követelmény analízis	Elvárások elemzése		
Specifikálás	Funkcionális és nem-funkcionális követelmények szisztematikus összegyűjtése, rögzítése	– Teljesség – Ellentmondás-mentesség – Ellenőrizhetőség	– **Statikus analízis** (felülvizsgálat vagy egyenrangú átvizsgálás) – Modell szimuláció
Architektúra tervezés	Specifikáció lebontása modul szintre (particionálás) Hardver és szoftver együttműködés kialakítása Kommunikáció	– Funkciók lefedése – Interfészek illeszkedése – Ütemezés – Balesetek, káresetek kockázata	– Statikus analízis – Szimuláció – Modell alapú analízis (teljesítmény, megbízhatóság) – **Veszély és kockázati analízis**
Modulok tervezése	Részletes belső működés megtervezése	– Kritikus algoritmusok, protokollok helyessége	– Statikus analízis – Formális verifikáció – Gyors prototípus
Modulok megvalósítása	Szoftver és hardver implementáció	– Biztonságos kód – Ellenőrizhető kód – Karbantartható kód	– Kódolási szabványok alapján történő ellenőrzés
Modul verifikáció	Modul működés igazolása	– Terveknek való megfelelés	– Statikus analízis – **Tesztelés (monitorozás)** – Regressziós tesztelés
Modul integrálás	Modulok illesztése, hardver-szoftver összeállítása	– Együttes működés megfelelősége	– **Integrációs tesztelés**
Rendszer verifikáció	Rendszerműködés igazolása	– Rendszer-specifikációnak való megfelelés	– **Rendszertesztelés (monitorozás)**
Rendszer validáció	Rendszer érvényesítése	– Követelményeknek, elvárásoknak való megfelelés	– Validációs tesztelés

3. Statikus analízis

Tervek, modellek, program forráskód végrehajtás nélküli elemzése:

- cél: hiányosságok felderítése
- módszer: „sorról sorra történő átolvasás” (szerző + átvizsgáló + teszt tervező + moderátor)
- ellenőrző listák használhatók
  - tipikus hibák esetén detektálást biztosítja
  - teljessége nem garantált, ezért hamis biztonságérzetet adhat
- automatizálás: pl. C nyelv esetén a *lint* program használható; jellegzetes hibák:
  - adathibák: inicializálás, értékhatárok,
  - vezérlési hibák: elérhetetlen kód
  - interfész hibák: típusok, számosság, visszatérési érték, értékadás nélküli output
  - tárkezelési hibák: felszabadítás

"Ceanroom" (tisztaszoba) fejlesztés: modul tesztelés helyett átvizsgálások

- formális specifikációból induló, helyességmegőrző transzformációk, statikus elemzés

4. A specifikáció teljességének és ellentmondás-mentességének vizsgálata

- Motiváció: Sok hiba visszavezethető a hiányos/ellentmondásos specifikációra
- Megoldás: szigorú specifikációs nyelv + utólagos ellenőrzés + tervezési minták használata

Az ellenőrzési lista alapelemei (ökölszabályok):

1. Az állapotdefiníció teljessége:
   - a kezdőállapot biztonságos;
   - belső modell aktualizálása megtörténik (ha egy adott ideig kimaradnak a bemeneti események, a belső modell egy *time-out* hatására aktualitását veszti);
2. Bemenetek (események) specifikációjának teljessége:
   - minden bemenetre (eseményre) van megadott reakció;
   - bemeneti ellenőrzés nem hiányzik (érték- és időtartomány);
   - megszakítások gyakorisága specifikált;
3. Kimenetek specifikációjának teljessége:
   - hihetőség-vizsgálat kritériumai specifikáltak;
   - fel nem használt kimeneteket ellenőrizni kell;
   - környezet feldolgozó kapacitását figyelembe kell venni (túlterhelés elkerülése);
4. Kimenetek és trigger bemenetek kapcsolata specifikációjának teljessége:
   - a kimenetek (hatásának) ellenőrzése szükséges bemeneteken (érzékelőkön) keresztül;
   - a szabályzási kör stabilitására ügyelni kell;

5. Állapotátmenetek specifikációjának teljessége:
   - minden állapot legyen elérhető;
   - állapotátmenetek (és a hozzájuk kapcsolódó kimenetek) visszafordíthatók legyenek;
   - veszélyes állapotból biztonságos állapotba több átmenet létezzen (ne ragadjon be);
   - biztonságos állapotból veszélyesbe csak jóváhagyással lehessen jutni (pl. egy érintkező hibája ne okozzon azonnal átmenetet a veszélyes állapotba);
   - katasztrofális állapot(konfiguráció) ne legyen elérhető;
6. Ember-gép interfész specifikációjának teljessége:
   - a kimenő események sorrendezése (prioritása), frissítése és időbelisége definiált legyen.

5. Veszély és kockázat analízis

- Biztonságkritikus rendszerek: A specifikáció nem teljesítése balesethez, káresethez vezethet.
- Tervezési és ellenőrzési folyamat specialitásai:
  - Szabványok, előírások, pl.
    - EN 50129 önálló vasútbiztonsági részrendszerek
    - DO178B repülőgép-fedélzeti beágyazott rendszerek
  - Folyamatos biztonsági analízis szükséges
  - Felelősségeket rögzíteni kell
  - Bizonylatolás (certification) szükséges

Terminológia:

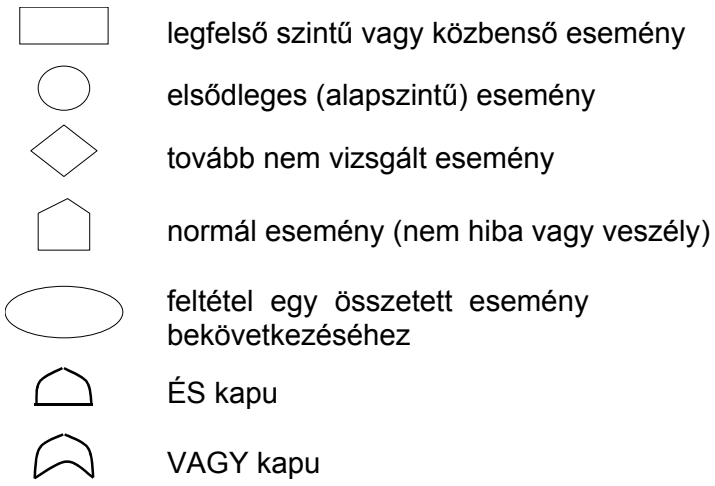
- Baleset: Nemkívánatos, be nem tervezett kár, veszteség
- Veszély, veszélyes állapot: Olyan állapot, amely adott környezeti feltételek mellett balesethez vezet
- Kockázat: Veszély szintje + időtartama + baleset valószínűsége
- Biztonságosság: Balesetektől való mentesség
- Biztonságkritikus szoftver: Veszélyes rendszerállapotokkal kapcsolatba hozható a szoftver hibamentes / hibás / hiányzó végrehajtása

Teendők a tervezés során:

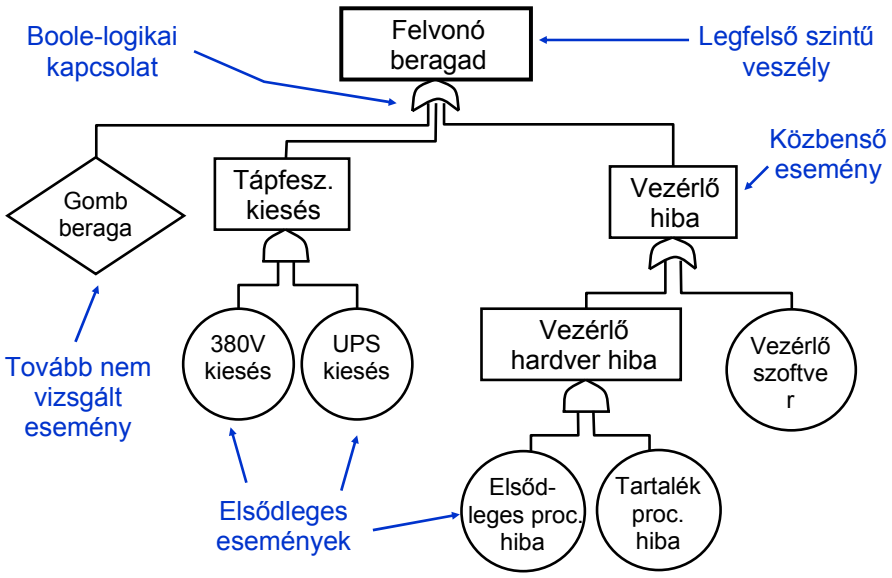
- Döntések ellenőrzése és dokumentálása
- Biztonságkritikus komponensek azonosítása
  Technika: Veszély analízis (ld. majd részletesen később!)
  Eredmény: Veszély katalógus: veszély azonosítása és jellemzése

5.1. Veszély analízis I.: Hibafa analízis

- Cél: Rendszerszintű veszély okainak szisztematikus vizsgálata
  - tipikusan felülről lefelé haladó analízis (a rendszerszintű veszélyből indul ki)
  - felderíti a kezelendő hibaokokat és -kombinációkat
- **Hibafa konstrukció:**
  - (rendszerszintű) veszély, veszélyes állapot: azonosítás a környezet, követelmények, szabványok alapján
  - közbenső események: veszélyhez vezetnek
  - pszeudo-események: alacsonyabb szintű események Boole-logikai kombinációi
  - elsődleges események: további felbontás nincs
- Hibafa grafikus elemkészlete



- Hibafa példa: Felvonó beragadása egy adott emeleten:



Kvalitatív analízis:

- Hibafa redukció: Közbenső események és pszeudo-események feloldása
  → diszjunktív normál forma
- Vágat: Egy AND kapuval összefogott elsődleges események
  Minimális vágathalmaz: Nem redukálható tovább
- Azonosítható:
  - egyszeres hibapont (SPOF): önmagában elég a rendszerszintű veszély kialakulásához
  - kritikus esemény (több vágatban is szerepel): kezelése hatékony a veszélyek csökkentése szempontjából

Kvantitatív analízis:

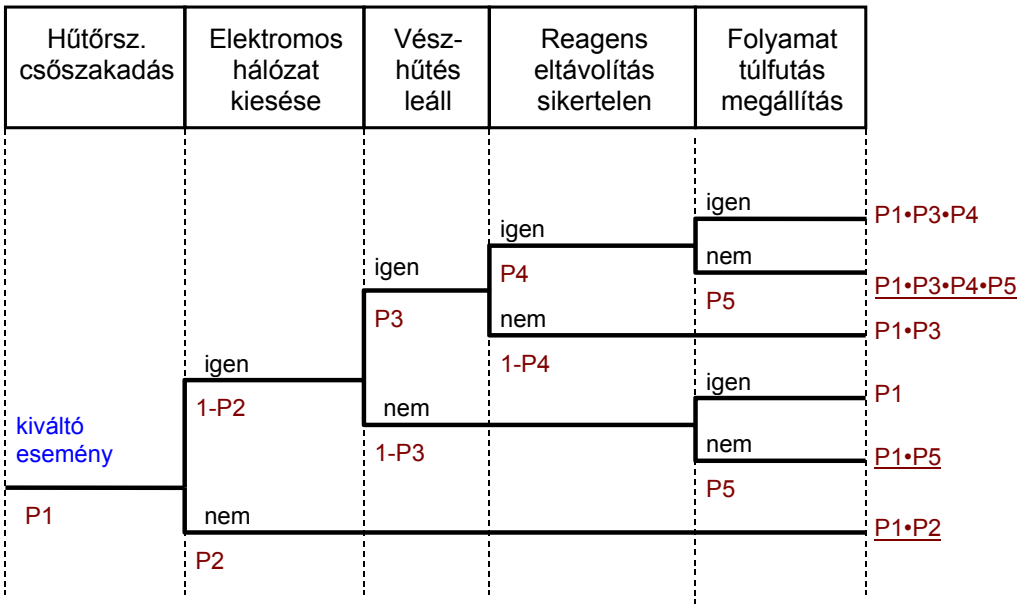
- Alapszintű eseményekhez rendelt valószínűségek
  - komponens-adat, tapasztalat, becslés
- Rendszerszintű veszély valószínűség számítása
  - AND kapu: szorzat
  - OR kapu: összegzés

  Feltétel: független események!
- Problémák:
  - több vágatban is szereplő esemény
  - korreláló hibák
  - időbeli (hiba)szekvenciák kezelése

5.2. Veszély analízis II.: Eseményfa analízis

- Előrelépő analízis: Elsődleges események következményeit vizsgálja
  - kiváltó esemény: pl. egy komponens hibája
  - következmények: más komponensek állapotától függ
  - sorrendezés: oksági kapcsolat, időbeli viszony
  - elágazások: események bekövetkezése
- Baleset „forgatókönyvek” vizsgálata
  - utak valószínűsége (elágazások valószínűsége alapján)
  - védelmi rendszerek hatékonysága
- Előnyök: Eseményszekvenciák vizsgálhatók
- Korlátok: Komplexitás, többszörös események, minden kiváltó eseményhez külön diagram

- Eseményfa példa: Hűtési rendszer egy vegyi folyamat esetén:



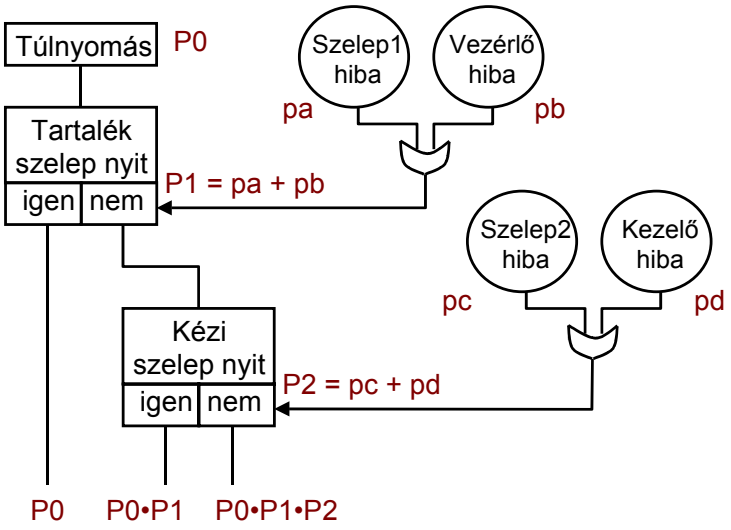
5.3. Veszély analízis III.: Ok-következmény analízis

Eseményfa és hibafa összekapcsolása
- eseményfa: forgatókönyvek (szekvencia)
- csatolt hibafa: esemény bekövetkezés „indoklása”, rendelkezésre állás számítása

Előnyök:
- szekvenciák (előrelépő analízis) és ok-okozati kapcsolatok (hátralépő analízis) együtt

Korlátok:
- minden kritikus eseményhez külön diagram szükséges

Példa:



5.4. Veszély analízis IV.: Hibamód és -hatás analízis

Módszer: Hibák és hatásaik (táblázatos) felsorolása

Előny:
- szisztematikus áttekintés
- redundancia felismerése

Komponens	**Hibamód**	**Valószínűség**	**Hatás**
L határérték-túllépés vizsgálat	> L átmegy ≤ L nem megy át	65% 35%	túlnyomás technológiai hiba
...	...	...	...

5.5. Veszély katalógus

Veszély analízis alapján a veszélyek besorolása:
- Veszély szint (MIL-STD-822b): katasztrofális, kritikus, mérsékelt, elhanyagolható
- Bekövetkezési valószínűség/gyakoriság: gyakori, valószínű, esetenkénti, ritka, valószínűtlen, lehetetlen

- Kockázati mátrix és védelmi szint (példa):

Veszély szint /gyakoriság	Katasztrofális	Kritikus	Mérsékelt	Elhanyagolható
Gyakori	P2 szelep beragad	.	.	.
Valószínű	.	Pumpa beragad	P3 szelep beragad	J1 jelfogó zárva beragad
Esetenkénti	Motor nem indul	.	.	.
Ritka	.	Tartály ereszt	.	D3 dióda szakadása
Valószínűtlen	Cső repedése	.	.	D3 dióda rövidzára
Lehetetlen	.	.	R1 ellenállás leég	.

5.6. Kockázatcsökkentés módszerei

- Veszély kiküszöbölés: elkerülni a veszélyt
  - helyettesítés: kevésbé veszélyes alkatrész, programnyelv
  - egyszerűsítés: determinisztikus, statikus vezérlési struktúra
  - szétcsatolás: modularizálás, jogosultságok kezelése
- Veszély csökkentés:
  - vezérelhetőség: inkrementális vezérlés, monitorozás
  - határolók: kizárás, bezárás, közrezárás
  - hiba minimalizálás: biztonsági tartomány
- Veszély kézbentartás:
  - időtartam csökkentés, elszigetelés, védőrendszerek
- Kár csökkentés:
  - menekülés, riadó tervek

6. Szoftver tesztelés

Tesztelés:

- Cél a program olyan futtatása, hogy a hibák kiderüljenek. Kimerítő tesztelés a gyakorlatban többnyire kivitelezhetetlen, ezért reprezentatív teszt eseteket kell meghatározni.
- Dijkstra: A tesztelés a hibák *jelenlétét*, és nem a hibamentességet tudja megmutatni!
- Hoare: Elméletileg a tesztelés egy *induktív bizonyítás* része: Ha a program egy adott (teszt)esetre jól működik, akkor várhatóan más esetekre is jól működik.

Alapfogalmak:
- Teszt: Bemenet és a hozzá tartozó helyes kimenet
- Egy hiba tesztje: Helyestől eltérő kimenetet kapunk
- Detektálható hiba: Van legalább egy tesztje
- Hibamodell: Azon hibák halmaza, amelyek felderítésére szolgál a tesztkészlet
- Hibafedés: A tesztkészlet által detektált hibák aránya az összes hibához képest

Tesztelés gyakorlati kérdései

1. Kivitelezhetőség, lehetőségek ismerete
   - Fejlesztési és futtatási környezet elkülönül. Verifikációs tesztelés a fejlesztési környezetben, validációs tesztelés a futtatási környezetben történik.
     - keresztfejlesztés
     - futtatási platform más
2. Hibamodell meghatározói a platform és a fejlesztési technológia
   - Platform hibái: Rendszerteszt és validációs teszt során figyelembe veendő
   - Fejlesztési technológia (pl. belső számítási modell) hibái: Modul tesztek a feltételezett hibák alapján konstruálhatók
   - Általános jellegű szoftver hibák:
     - algoritmus hibák ← (formális) verifikáció vizsgálja
     - kódolási hibák: határérték, inicializálás, mutatók, vezérlési folyamat,... (részben statikus ellenőrzéssel vizsgálható) ← automatikus kódgenerálás kiküszöböli
     - teljesítmény / időzítés hibák
     - felhasználói interfész hiányosságai
3. Szabvány előírások: Megfelelő minőségű tesztelés (felhasználó nem lehet "béta-teszter").

6.1. Teszttervezés megközelítései

1. Funkcionális tesztelés
- a rendszer mint „fekete doboz” adott
- csak a külső viselkedés (funkció) ismert, a belső felépítés nem
- tesztelés alapja: specifikált funkciók megléte, extra funkciók hiánya

2. Strukturális tesztelés
- a rendszer mint „üvegdoboz” adott
- a belső struktúra is ismert
- tesztelés alapja: belső működés végigkövetése: vezérlési folyamat-gráf, vezérlési gráf

6.1.1. Funkcionális tesztelési módszerek

1. Ekvivalencia particionálás

Meggondolások:
- Egy teszt minél több bemeneti feltételt írjon elő (minél több hibát felfedhessen)
- Egy teszt minél több bemenettel legyen ekvivalens (minél kevesebb tesztet hajtsunk végre)

Bemenet ekvivalencia osztályai:
Bemeneti adatok, amelyek várhatóan ugyanazon hibát fedik le
pl. adott hőmérséklet tartományokban hűt / kikapcsol / fűt egy rendszer; ezek a tartományok ekvivalencia osztályokat képeznek
Cél: Egy ekvivalencia osztályból egy teszt elég legyen (az adott bemenethez)

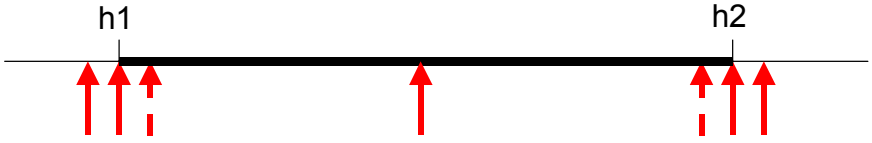
Ekvivalencia osztályok meghatározása:
Heurisztikus folyamat; példák:
- érvényes és érvénytelen bemeneti adatok
- értékhatárokon belüli, alul-felül túllépő adatok

Tesztek meghatározása több bemenet esetén:
- érvényes ekvivalencia osztályok: egy teszt minél több osztályt fedjen le
- érvénytelen ekvivalencia osztályok: minden osztályhoz külön teszt legyen (egymás hatását ne oltsák ki)

2. Határérték-analízis

Vizsgálandó a bemeneti és kimeneti adattartományok alsó és felső határa
- egy ekvivalencia osztály határaira koncentrál
- kimeneti értékeket is figyelembe veszi

Tipikus adatok:
- egy határérték 3 tesztet jelent
- egy tartomány 5-7 tesztet jelent



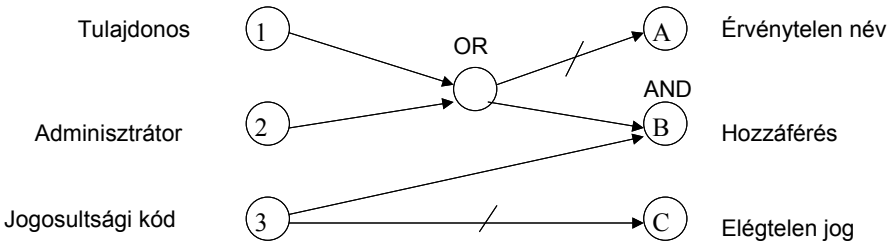
3. Ok-hatás analízis

A bemeneti feltételek kombinációinak vizsgálata
- Ok: egy-egy bemeneti ekvivalencia osztály
- Hatás: egy-egy kimeneti feltétel (ekvivalencia-osztály)

Boole-gráf: Okok és hatások összekapcsolása
- ÉS, VAGY kapcsolatok
- meg nem engedett kombinációk

Döntési táblázat: A gráf szisztematikus végigjárása (logikai hálózat igazságtáblázata)
- egy oszlop egy tesztnek felel meg

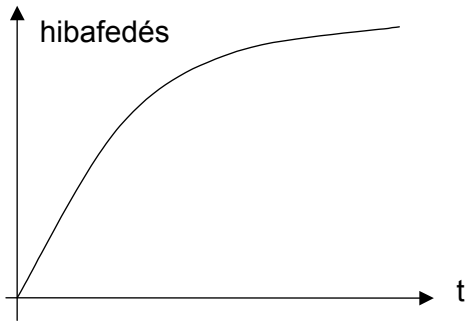
Példa: Hozzáférés ellenőrzés



	T1	T2	T3
1	0	1	0
2	1	0	0
3	1	1	1
A	0	0	1
B	1	1	0
C	0	0	0

Módszerek együttes alkalmazása:
1. Ekvivalencia particionálás
2. Határérték-analízis
3. Ok-hatás analízis

Tesztek ütemezése: A várhatóan nagyobb hibafedésű teszteket célszerű először végrehajtani.
A hibafedés változása a tesztelési idő függvényében hatékony tesztelés esetén:



Kiegészítés: Véletlen tesztelés
- véletlen tesztvektorok generálása
- kis számítási teljesítményt igényel, gyors, de hibafedése kicsi
- referencia: válasz számítása, szimulálása vagy csak “elfogadhatósági vizsgálat”

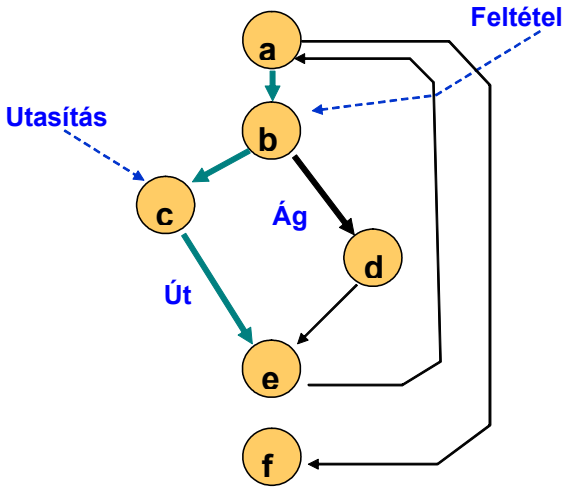
6.1.2. Strukturális tesztelési módszerek

- Általános jellemzők:
  - Vezérlési szerkezet ismert, ennek minden ágát meg kellene vizsgálni
  - Hibamodell implicit ("felsorolás" nincs): Minden hiba, ami a vezérlési szerkezetet érinti (feltételek, ugrási címek, ki/belépési pontok,...)
- Előnyök:
  - A beágyazott rendszerekben tipikusan domináns a vezérlés, ezt kell tehát tesztelni
  - Jól kezelhető modell létezik: a vezérlési gráf
  - Tesztelés hatékonysága számszerűen jellemezhető (vezérlési gráf lefedése)
    De ez nem a hibafedés!
  - Szabványok előírása vezérlés-intenzív alkalmazásokra: pl. DO-178B
- A vezérlési gráf (példa):

Forráskód:

```
a:  for (i=0; i<MAX; i++) {
b:      if (i==a) {
c:          n=n-i;
        } else {
d:          m=n-i;
        }
e:      printf("%d\n",n);
    }
f:  printf("Ready.");
```

Vezérlési gráf:



- Vezérlési gráf lefedése (végrehajtása) tesztekkel:
  - Utasítások lefedése: Minél több utasítást végrehajtani
  - Döntési ágak lefedése: Elágazások esetén minden irányban továbbmenni
  - Feltételek lefedése: Feltételes elágazásokban minél több feltétel kombinációt kipróbálni
  - Utak lefedése: Minél több független utat bejárni a gráfban
- Tesztminőségi mértékszámok rendelhetők a teszt készlethez:
  A tesztelhető (lefedhető) elemek mekkora részét teszteltük:
  1. Utasítás lefedettség
  2. Döntési ág lefedettség
  3. Feltétel lefedettség
  4. Út lefedettség

1. Utasítás lefedettség:

$$\frac{\text{Tesztelés során végrehajtott utasítások száma}}{\text{Összes utasítás száma}}$$

Nem szigorú:
Utasítások kihagyási feltételeit nem veszi figyelembe
pl. `k=0; if (a>0) k=1; m=1/k;` lefedése esetén `az a=0` eset vizsgálata kimarad

2. Döntési ág lefedettség:

$$\frac{\text{Tesztelés során végrehajtott döntési ágak száma}}{\text{Összes lehetséges döntési ág száma}}$$

Nem szigorú:

```
if (a>0 && (safe(c) || safe(b)) start();
else stop();
```

A fordító optimalizálása miatt a safe(b) hiívás be sem következik, ha safe(c) igaz.

3. Feltétel lefedettség:

$$\frac{\text{Feltételekben a tesztelt bemeneti kombinációk száma}}{\text{Feltételek összes bemeneti kombinációinak száma}}$$

Aránylag bonyolult tesztelés a sok lehetséges kombináció beállítása miatt.

4. Út lefedettség:

$$\frac{\text{Bejárt független utak száma}}{\text{Összes független út száma}}$$

Független utak: van olyan pont vagy él, ami a másikban nincs meg
Jellemzés: A gráf ciklomatikus komplexitása (CK): A független utak *maximális* száma
Meghatározás:
CK(G)=E-N+2, ahol
E: élek száma
N: pontok száma a G vezérlési gráfban
100% út lefedettség együtt jár:
– 100% utasítás lefedettség
– 100% ág lefedettség

- Tesztelési módszer:
  - Tipikus cél: Független utak lefedése (a független utak halmaza nem egyedi!)
  - Egyszerű algoritmus:
    - CK számú független út kiválasztása
    - bemenetek generálása egy-egy út bejárásához
  - Problémák, nehézségek:
    - ciklusok: korlátozni (minimalizálni) kell a bejárást
    - generálható-e a bejáráshoz bemeneti szekvencia (nem minden út bejárható)
    - lehetséges-e a belső változók beállítása

6.2. Tesztelési folyamatok áttekintése

1. Modultesztelés
- izolációs tesztelés

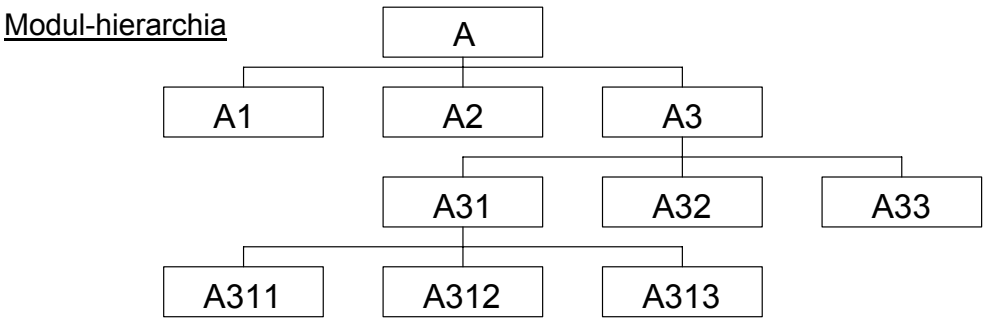
2. Integrációs tesztelés
- top-down (felülről-lefelé) tesztelés
- bottom-up (alulról-felfelé) tesztelés

3. Rendszertesztelés

4. Validációs tesztelés
- időzítési szempontok
- környezeti szimuláció

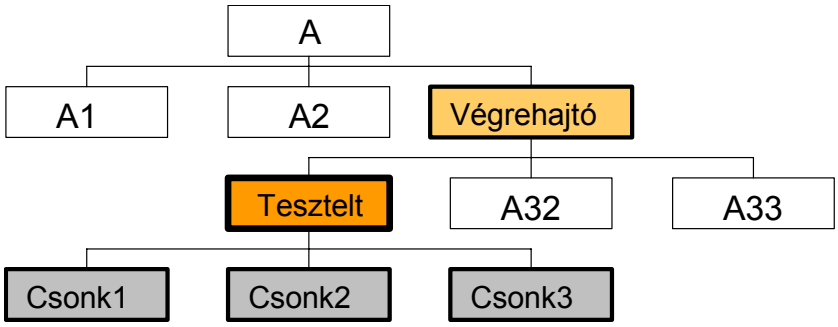
6.2.1. Modultesztelés

- Modultesztelés előnyei:
  - integrációs fázis hatékonyabb, ha a modulok már teszteltek
  - integráció után nehéz egy-egy modult alaposan tesztelni
  - minél hamarabb felderíthető egy hiba, annál olcsóbb a tesztelése
  - párhuzamosítható folyamat
- A modultesztelés tipikusan strukturális tesztelés
  - (funkcionális tesztelés a rendszerszinthez illeszkedik)
  - segíti a fejlesztést (a modul kódolói végzik)
- Regressziós tesztelés:
  - a modul minden változtatásakor
  - környezet változásakor (tesztkörnyezet is)
- A modulok hierarchiába rendezhetők egymás szolgáltatásainak használata (hívások) alapján. A tesztelés során figyelembe kell vennünk ezt a modulhierarchiát is.



Izolációs tesztelés

- modulok egyenként, elszigetelten teszteltek
- tesztvégrehajtó (vezérlés és megfigyelés biztosítása) és teszt csonk (hívott modul egyszerű helyettesítése) szükséges
- korai integrációt nem biztosít
- módosítás csak egy modul újratesztelését jelenti



6.2.2. Integrációs tesztelés

Kapcsolódási **interfészek** tesztelése:

- A rendszer annak ellenére hibás lehet, hogy minden modul egyenként hibátlan!

Módszerek:

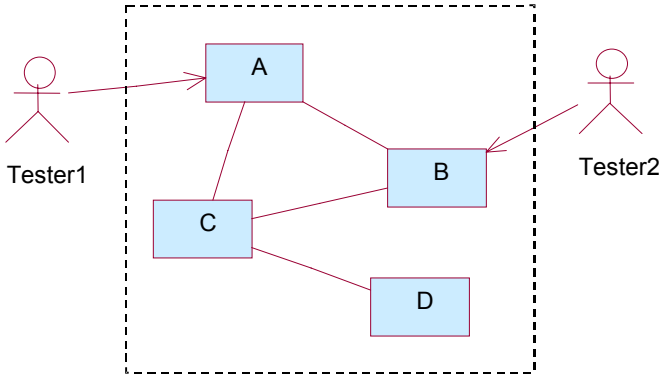
- Funkcionális tesztelés: Együttműködési scenariok tesztje
- (Strukturális tesztelés csak modulszinten!)

Megközelítés:

- "Big bang": minden modult egyszerre integrálni
- Inkrementális: egyenként összerakni

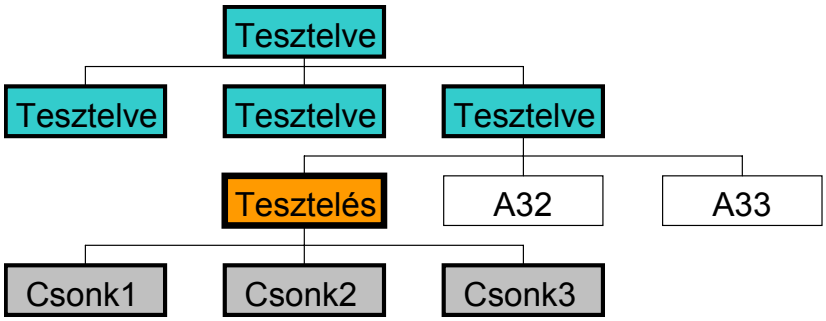
1. "Big bang" tesztelés

- Tesztelés a külső interfészeken keresztül
- Külső teszt végrehajtó
- Rendszer funkcionális specifikáció alapján történik
- Modul módosítás: Újratesztelés
- Kis rendszerek esetén alkalmazható: Belső modulokban nehéz a hibakeresés!



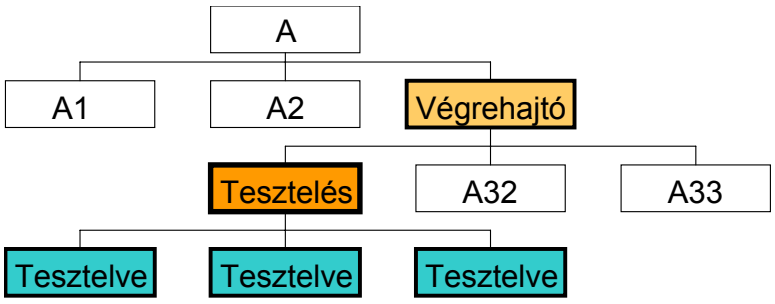
2. Top-down tesztelés

- modulok a már tesztelt hívó modulokból kerülnek tesztelésre
- csonkok szükségesek (később ezek helyébe tesztelendő modulok lépnek)
- erősen követelmény-orientált (“fentről” tesztelünk)
- modul módosítás: alatta lévők tesztelését módosítja



3. Bottom-up tesztelés

- tesztelendő modul a már teszteltet használja
- tesztvégrehajtó szükséges (később ennek helyébe a tesztelt modul lép)
- korai integrációt biztosít
- modul módosítás: felette lévők tesztjére hatással van



4. Futtató rendszer integrációja

Motiváció: Nehéz csonkokat írni a futtató rendszerhez (OS, RT-OS, taszk ütemezés ...)
Módszer:

1. Alkalmazás modulok integrációja felülről lefelé, a futtató rendszer szintjéig
2. Futtató rendszer alulról felfelé történő tesztelése
   - komponensek izolációs tesztje
   - „big bang” tesztelés teszt végrehajtóval
   - „big bang” tesztelés az alkalmazás hierarchia legalsó rétegével
3. Alkalmazás és futtatórendszer integrációja, a felülről lefelé történő tesztelés befejezése

Eszközök az integrációs teszteléshez: Wrapper (csomagoló) kódrészletek

- „before” wrapper: A hívás végrehajtása előtt
  - paraméterek vizsgálata
  - hívási szekvencia ellenőrzése
- „after” wrapper: A hívás végrehajtása után
  - visszaadott érték ellenőrzése vagy módosítása
- „replace” wrapper: A hívott helyettesítése
  - teszt csonk megvalósítás

6.2.3. Rendszertesztelés

Tesztelés a **rendszerszintű specifikáció** alapján

- Típusok (többek között):
  - szolgáltatás tesztelés
  - mennyiségi tesztelés
  - stressz (löket) tesztelés
  - használhatósági tesztelés
  - biztonsági tesztelés
  - teljesítmény-tesztelés
  - konfiguráció tesztelés
  - megbízhatósági tesztelés
  - dokumentáció tesztelés, ...

6.2.4. Validációs tesztelés

- Cél: **Valóságos környezet** hatásának tesztelése
  - váratlan eseményekre való reagálás
  - kis valószínűségű eseménykombinációk is
  - túlterhelés / alulterhelés vizsgálata
- Időzítési szempontok
  - valós időkorlátok meghatározása adott környezetben
  - korlátok ellenőrzése

  → valósidejű monitorozás (ld. következő fejezet)
- Környezeti szimuláció
  - adott szituációk valóságban nem tesztelhetők (pl. védelmi rendszerek)
  - kérdéses, a szimulátorok mennyire valósághűek.

7. Monitorozás

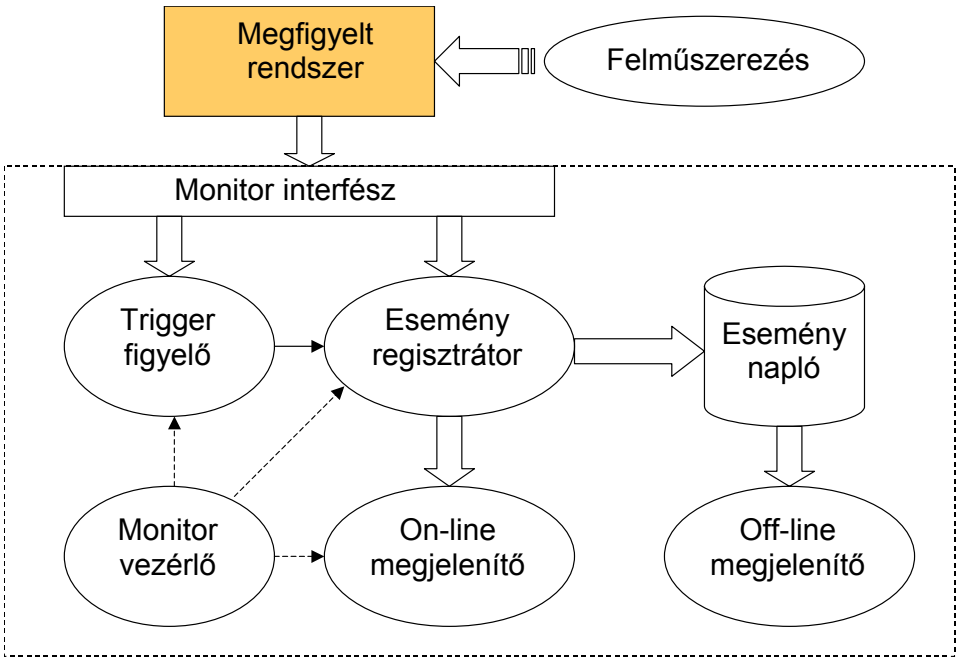
- Cél: Futásidőbeli információk begyűjtése
  Használható:
  - Verifikációs tesztelés: pl. előírások ellenőrzése (teszt végrehajtó része)
    Hiba azonosítás (diagnosztika, debuggolás) segítése
  - Validációs tesztelés: pl. teljesítmény elemzés valós környezetben
    Feltételek:
    - prototípus + futtató környezet rendelkezésre áll
    - várható terhelés (benchmark) ismert
  - Működés közben: biztonsági ellenőrzés (pl. biztonsági monitor)

Alapműveletek a monitorozáshoz:

- Információ hozzáférés: Felműszerezés
  - szoftver úton: extra utasítások beszúrása a kódba -- viselkedés változik!
  - hardver úton: csatlakozás a rendszerbuszra, processzor lábakra, kimenetekre
- Információ szűrés: Triggerelés (a megfigyelni kívánt feltételek teljesülésére várni)
  - szoftver úton: extra utasítások megoldják
  - hardver úton: jelek, busz forgalom figyelése
- Információ tárolás: Regisztrálás (a megfigyelni kívánt állapot feljegyzése)
  - szoftver úton: naplózás szoftver modulból
  - hardver úton: jelek mintavételezése („logikai analizátor”)

Általános monitorozási séma

Szükséges monitor interfész:
- esemény detektálás
- esemény rögzítés
- beállítások (pl. trigger esemény megadása)



Nehézségek, problémák a monitorozás során:

- Beavatkozás: Ha a rendszer erőforrásait használjuk, (ld. triggerelés, regisztrálás) akkor megváltoztatjuk a rendszer viselkedését
  - időviszonyok → eltérő ütemezés
  - esemény sorrend → akár holtpont (deadlock) is kialakulhat / elfedődik

  Megoldás:
  - hardver monitorozás (legkevésbé avatkozik be)
  - zavarás (perturbáció) korrekciója:
    - időzítések korrigálása
    - esemény-szekvencia módosítása?
  - monitorozó kód bennhagyása a végleges rendszerben

- Szemantikai hézag: Megfigyelt információból a számunkra érdekes származtatása
  - pl. processzor buszjelekből a szemafor műveletekre következtetni...

  Megoldás:
  - szoftver monitorozás vagy bonyolult off-line kiértékelés

- Globális jellemzők: Elosztott rendszerekben a lokálisan gyűjtött információkból a globális állapot meghatározása (pl. körkörös várakozás detektálása)

  Megoldás:
  - központi monitor használata dedikált kommunikációs csatornákkal
  - elosztott monitorok szinkronizálása (globális óra, időbélyegek, esemény-sorrendezés)

Célkitűzések

- Transzparens monitorozás (alkalmazás-független, programozói közreműködés nélkül)
- Minimális beavatkozás, vagy korrekció (időzítés, eseménysorrend)
- Minimális hardverfüggőség (szabványos interfész buszrendszerekhez)
- Minimális erőforrásigény (regisztráláshoz, eseményszűréshez); konkurens futás

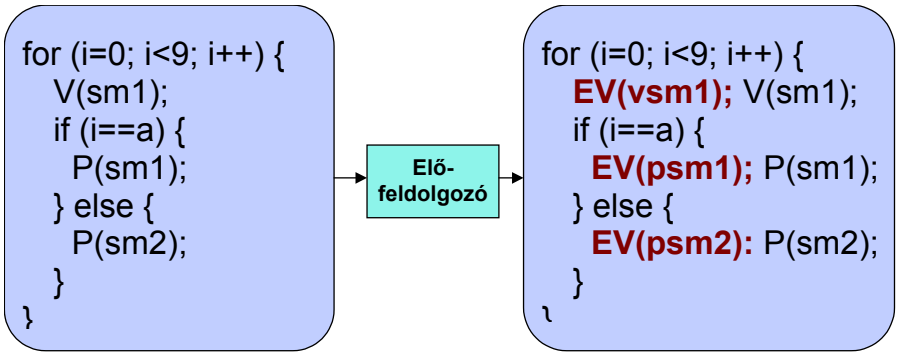
Megközelítések áttekintése

1. Szoftver alapú monitorozás
   - mindent szoftverből oldunk meg
   - rugalmas, de beavatkozó
2. Hardver alapú monitorozás
   - triggerelés (és regisztrálás) hardver egységgel
   - nem beavatkozó, de drága és specifikus
3. Hibrid monitorozás
   - triggerelés szoftver (rugalmas), regisztrálás hardver (kevésbé beavatkozó) egységgel
   - kompromisszumos megoldás

7.1. Szoftver alapú monitorozás

Megfigyelt rendszerbe beágyazott monitor

- Felműszerezés: Trigger utasítások beszúrásának módja:
  - kézi: csak az „érdekes” eseményekre
  - automatikus: esemény-osztályokra (pl. szemafor műveletek);
    ld. aspektus-orientált programozás: beavatkozási pontok kijelölése



Felműszerezés: Trigger utasítások helye (láthatóság):
- kernel: ütemezés is megfigyelhető, alkalmazás-független (de kernel forrás szükséges)
- alkalmazás: kimenet vizsgálható, magasabb absztrakciós szintű lehet

Beavatkozás kezelése:
- trigger utasítások a rendszerben maradnak
  ha nem hoznak be indirekt függőséget (pl. regiszter módosítás)
- regisztráláshoz szükséges kód leválasztható

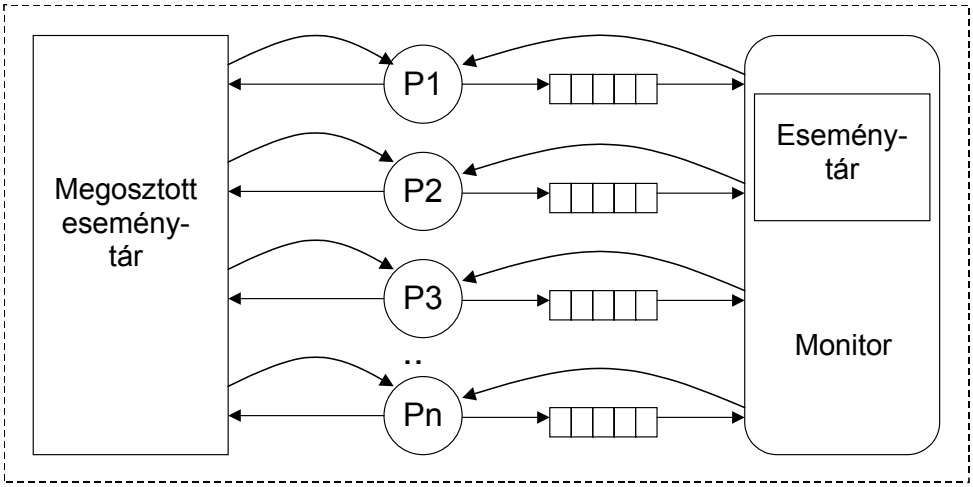
- Triggerelés: Trigger utasítások jeleznek
  - alkalmazás felé
  - kernel felé
- Regisztrálás: Események feljegyzése
  - alkalmazás végzi: közös memória a megfigyelt rendszerrel
  - általános monitor processz végzi: leválasztva az alkalmazásról

Tipikus megközelítések:

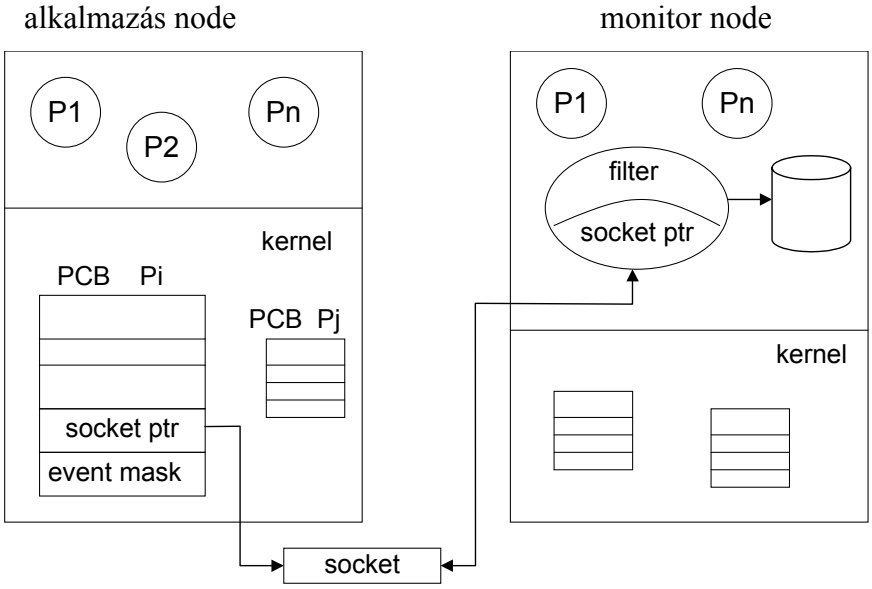
Triggerelés	**Regisztrálás**	**Jellemzők**
alkalmazás	alkalmazás	– rugalmas, magas szintű triggerelés – alkalmazással közös memória
alkalmazás	monitor processz	– rugalmas, magas szintű triggerelés – leválasztott monitor memória, külön ütemezés
kernel	monitor processz	– alacsony szintű triggerelés, de kernel szintű események is megfigyelhetők – leválasztott monitor memória, külön ütemezés

Példa: IBM RS/6000

- Szinkron monitor: Megosztott tárolás, azonnali ellenőrzés
- Aszinkron monitor: Eseménysorok egy monitor processzhez



Példa: Berkeley Unix elosztott monitor: PCB kiegészítése, maszkolás, szűrés



Példa: Profilerek

- Használat:
  - függvényhívási hierarchia felderítése, hívási gyakoriság hozzárendelése
  - az egyes függvényekben eltöltött idő hozzárendelése
    → mit érdemes újraírni, optimalizálni
  - annotált forrás: programsorok végrehajtási gyakorisága
  - célkörnyezetben futtatva használhatók az eredmények
- pl. gprof
  - program felműszerezése: speciális fordítás
    cc -pg prog.c -o prog (opció; gcrt0.o, libc_p.a)
  - futás közbeni adatgyűjtés: adat file-ok (gmon.out)
  - off-line kiértékelés: gprof prog gmon.out

7.2. Hardver monitorozás

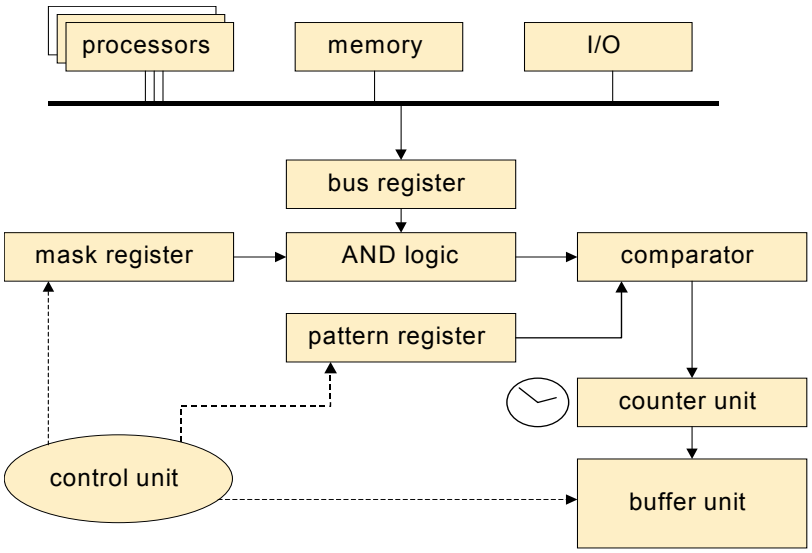
Megfigyelt rendszertől független monitorozás: → szigorúan RT rendszerekben is használható

- Felműszerezés: Csatlakozás busz vezetékekhez
- Triggerelés: Hardver trigger figyelő
  - trigger események: vezérlőjelek, adat / címbusz "mintaillesztés"
    utasításkód, címtartomány, megszakítás
  - vezérlő: trigger események megadása
- Regisztrálás: Busz monitor
  - lokális: puffer (pl. cirkuláris puffer)
  - elosztott rendszer: globális állapot rögzítése
    dedikált csatornák a lokális monitorok szinkronizálására
- Megközelítések:
  - „phantom memory": a monitorozott rendszer memóriájának tükrözése (snapshot)
  - mintaillesztés a monitorozott rendszer buszjeleire:
    minden vizsgálandó információ a buszra kell kerüljön (pipeline, belső cache)
  - mintaillesztés eseményekre:
    - magasabb szintű megfigyelés
    - időadatok tárolandók
    - nincs információ alacsony szintű hibakereséshez
  - monitor illesztése dedikált kommunikációs hálózaton
    - lokális eseménydetektálás, értesítés a hálózaton
    - eseményszámlálás (kevés információ)

Mért adatok értelmezése:

- Hardver egységek: közvetlen használható jelek
  - busz hozzáférés, foglaltság; cache találatok, feltöltés; I/O műveletek, I/O időzítés
- Szoftver komponensek: buszjeleket értelmezni kell
  - utasításkódok kiszűrése
  - párhuzamos végrehajtás látható
  - szűk keresztmetszetek azonosítása: közös memóriaterületre való írás stb.

Példa: Többprocesszoros rendszer

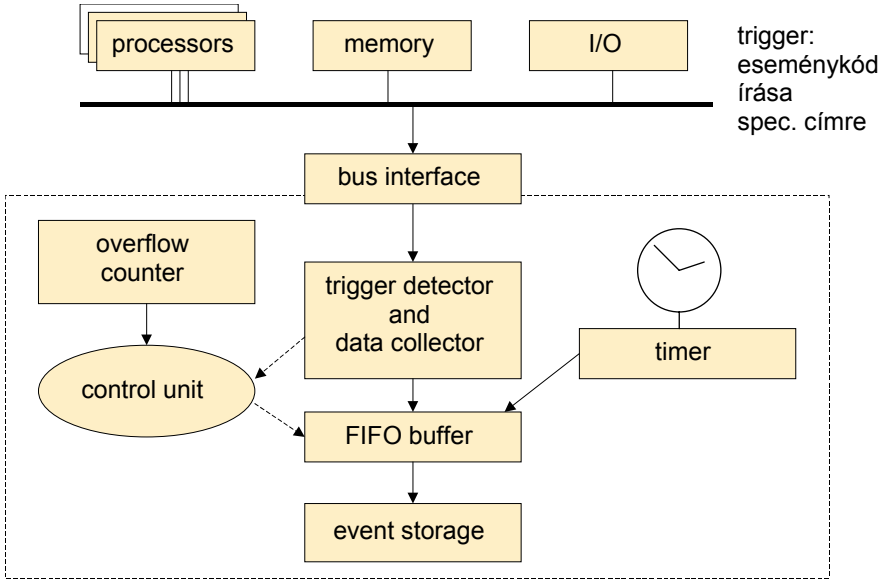


7.3. Hibrid monitorozás

Beavatkozás minimalizálása + rugalmasság

- Felműszerezés: Trigger utasítások beszúrása
  - rugalmas, magasabb szintű lehet
  - kis overhead: akár 1%-ra csökkenthető
- Triggerelés: Szoftver
  - alkalmazás / kernel szintű
- Regisztrálás: Hardver monitor
  - lokális: koprocesszor
  - elosztott rendszer: a monitor is elosztott lesz

Példa: Többprocesszoros rendszer



7.4. Hibakeresés monitor segítséggel

Monitorozással megfigyelhető tipikus hibák

- Végrehajtás (számítás) időkorlátai:
  - túlságosan hosszú végrehajtási idő (time-out)
- Kommunikáció hibái: ← üzenetek
  - elvesztett kérés / válasz / nyugta
  - üzenet többszörözés
  - szerver meghibásodás: kliens időtúllépés
  - kliens meghibásodás: árva üzenet a szerveren
  - helyreállítás: duplikált üzenetek
- Szinkronizáció hibái: ← üzenetek
  - globális óraszinkronizáció
  - kiéheztetés (kölcsönös kizárás)

- Ütemezés hibái: ← időbélyegek
  - túl kevés ütemezett idő / túl nagy késleltetés
- Holtpont: Várakozás
  - folyamatra ← szinkronizáció monitorozása
  - üzenetre ← kommunikáció monitorozása
  - erőforrásra ← foglalás / elengedés monitorozása

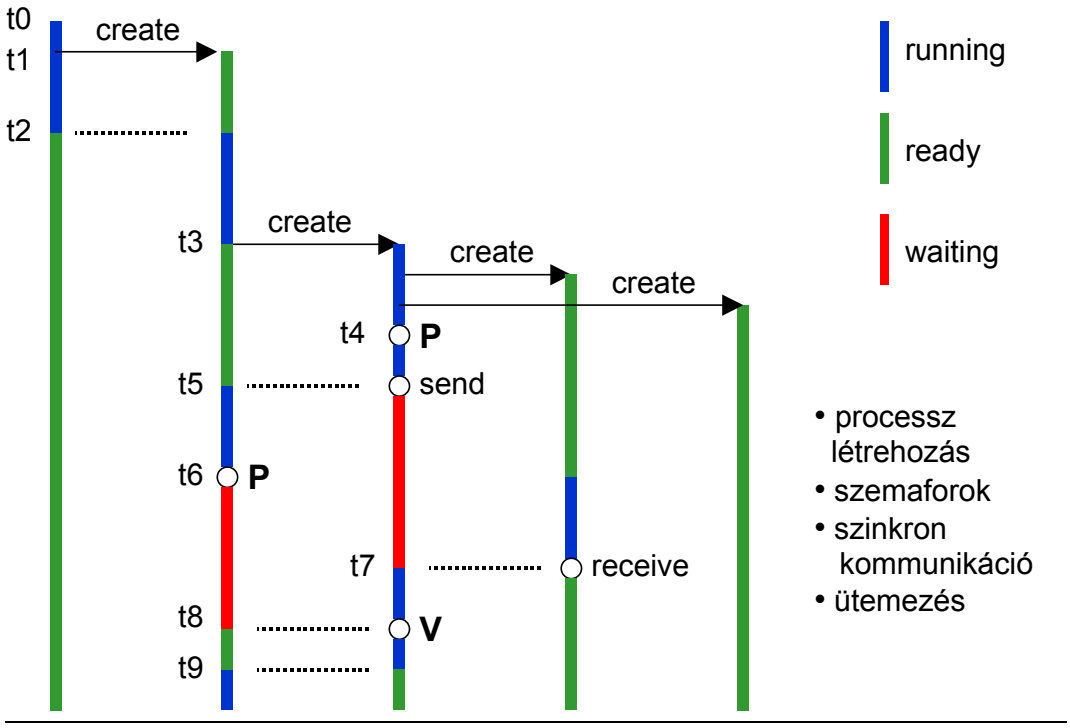
Hibakeresés technikái

Cél: Detektálás, lokalizálás, izoláció, javítás

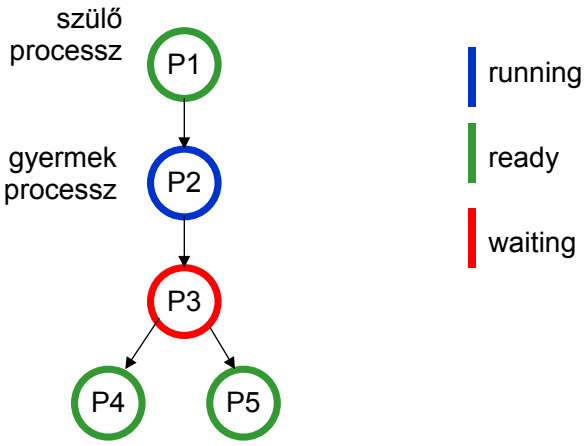
- Valósidejű megjelenítés
  - kevés információ áttekinthető, felfüggeszteni nem lehet
- Valósidejű hibakeresés
  - monitorba kell építeni a hibadetektálást (pl. time-out)
- Interaktív hibakeresés (ha megállítható a rendszer)
  - valósidejű megjelenítés és beavatkozás
- Determinisztikus visszajátszás
  - off-line visszajátszás és elemzés (részletgazdag naplózás szükséges)
- Dinamikus szimuláció
  - újrafuttatás (más teszt adatokkal)

A monitorozott rendszerjellemzők grafikus megjelenítése

- **Színezett processz együttműködési gráf**
  - szülő-gyermek folyamatok megfigyelhetők
  - szinkronizáció bejelölve
  - dedikált (adott időkorlátra vonatkozó) vagy szűrt (adott eseményre vonatkozó)



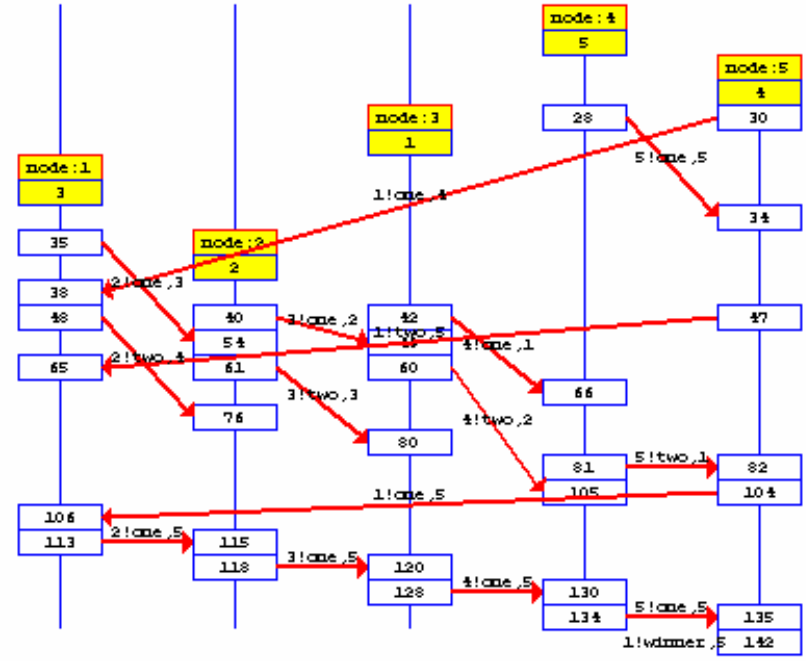
- **Processz státusz gráf**:
  - szülő / gyermek processzek, színnel jelölt ütemezési információ



- **Szemafor allokációs mátrix vagy gráf**
  - szemaforok (S1, S2, S3, ...) és processzek (P1, P2, ...) kapcsolata (P / V)

	P1	**P2**	**P3**	**P4**	**P5**
S1		+	-		
S2			+	-	-
S3	+	-			

- **Üzenetküldési mátrix vagy gráf**
  - üzenetek és processzek kapcsolata (küldés, fogadás)
  - (üzenet) szekvencia diagram



8. Hibatűrés

- Akármilyen jó is az ellenőrzés, a szolgáltatásbiztonság nem garantálható
  - időleges hardver hibák (ld. zavarérzékenység)
  - teszteletlen szoftver hibák
  - figyelembe nem vett komplex interakciók

  → Fel kell készülni a működés közbeni hibákra!

- Hibatűrés: Szolgáltatást nyújtani hiba esetén is
  - működés közbeni autonóm hibakezelés
  - beavatkozás a meghibásodás → hibajelenség láncba
  - rendszertechnikai megoldások (+ megbízható alkatrészek)

- Alapfeltétel: Redundancia (tartalékolás)
  - többlet erőforrások a hibás komponensek kiváltására

Redundancia megjelenése

1. Hardver redundancia
   - többlet hardver erőforrások
     - eleve a rendszerben lévők (elosztott rendszer)
     - hibatűréshez betervezett (tartalék)
2. Szoftver redundancia
   - többlet szoftver modulok
3. Információ redundancia
   - többlet információ a hibajavítás érdekében
     - hibajavító kódolás (ECC)
4. Idő redundancia
   - ismételt végrehajtás, hibakezelés többlet ideje

   Együttes megjelenés!

Redundancia típusai

- Hidegtartalék (passzív redundancia):
  - normál üzemmódban passzív, hiba esetén aktiválva
  - lassú átkapcsolás (elindítás, állapot frissítés,...)
  - pl. tartalék számítógép

- Langyos tartalék:
  - normál üzemmódban másodlagos funkciók
  - gyorsabb átkapcsolás (indítást nem kell várni)
  - pl. naplózó gép átveszi a kritikus funkciókat

- Meleg tartalék (aktív redundancia):
  - normál üzemmódban aktív, ugyanazt a feladatot végzi
  - azonnal átkapcsolható
  - pl. kettőzés, többszörözés

Hibák kezelése

- Hardver tervezési hibák (< 1%):
  - nincs figyelembe véve (ld. jól tesztelt komponensek)
- Hardver állandósult működési hibák (10%):
  - hardver redundancia (pl. tartalék processzor)
- Hardver időleges működési hibák (70-80%):
  - idő redundancia (pl. utasítás újravégrehajtás)
  - információ redundancia (pl. hibajavítás)
  - szoftver redundancia (pl. állapotmentés és helyreállítás)
- Szoftver tervezési hibák (10-20%):
  - szoftver redundancia (pl. eltérő tervezésű modulok)

8.1. Állandósult hardver hibák kezelése

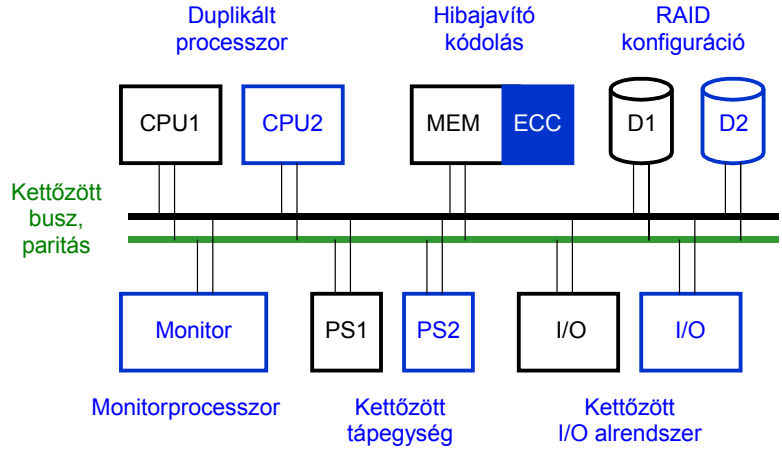
Többszörözés:

- Kettőzés
  - hibadetektálás: pl. master-checker kialakítás (ld. Pentium processzor)
  - hibatűrés csak diagnosztikai támogatással és átkapcsolással
- TMR: Triple-modular redundancy
  - hiba maszkolás szavazással
  - szavazó kritikus elem (de egyszerű)
- NMR: N-modular redundancy
  - hibamaszkolás többségi szavazással
  - MTFF kisebb, de missziós idő túlélése nagyobb esélyű
  - repülőgép fedélzeti eszközök: 4MR, 5MR

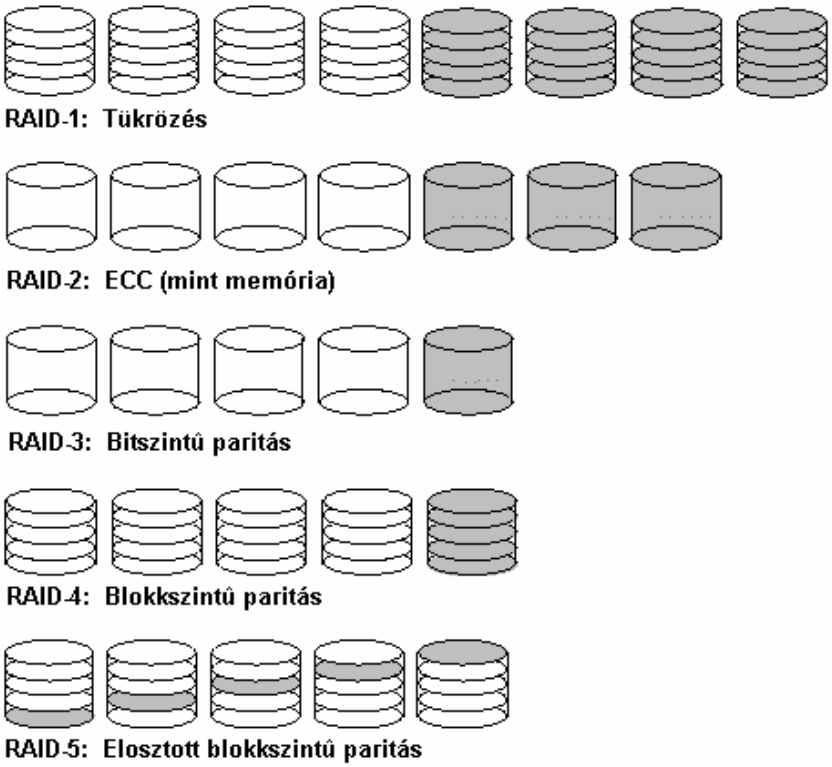
Többszörözés szintje:

- Számítógép (szerver) szint: Lazán csatolt
  - nagy rendelkezésreállású klaszterek, pl. Sun HA cluster, HA Linux
  - szoftver támogatás: állapotszinkronizálás, tranzakciók
- Kártya szint:
  - futásidőbeli átkonfigurálás “hot-swap”; pl. compactPCI, HDD
  - szoftver támogatás: konfigurációkezelés
- Alkatrész szint: Szorosan csatolt
  - alkatrész szintű többszörözés, pl. TMR, önellenőrző áramkörök

Áttekintés hardver redundanciáról tipikus számítógépes rendszerekben:



RAID (Redundant Array of Independent Disks)

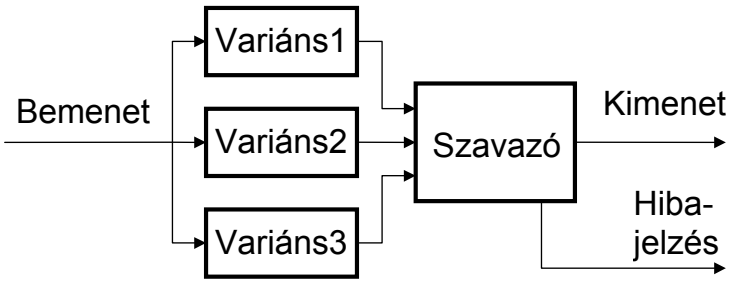


8.2. Szoftver tervezési hibák kezelése

- Ismételt végrehajtás nem segít (állandósult hiba)
  Redundáns modulok: eltérő tervezés szükséges:
  variánsok: azonos specifikáció, de
  - eltérő algoritmus, adatstruktúrák
  - más fejlesztési környezet, programnyelv
  - elszigetelt fejlesztés
- Variánsok végrehajtása:
  – N-verziós programozás
  – javító blokkok (recovery block)

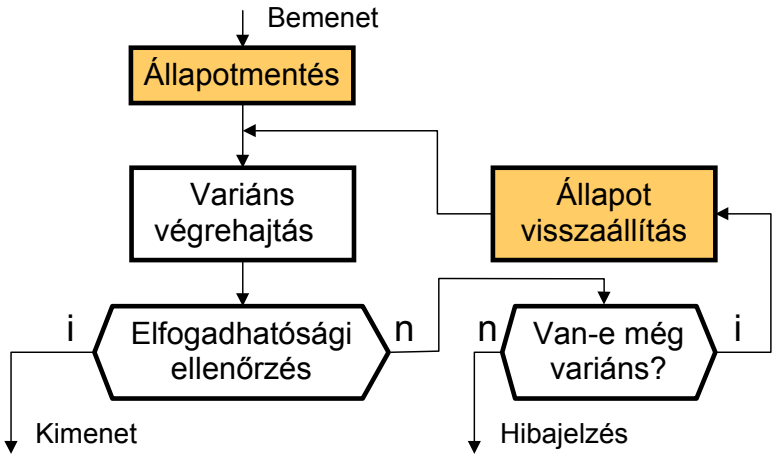
N-verziós programozás

- Aktív redundancia: párhuzamosan végrehajtható
  - ugyanazon bemenetek
  - szavazás (SPOF): megengedhető tartomány



Javító blokkok

- Passzív redundancia: Csak hiba esetén aktiválódik
  - variánsok kimenetének elfogadhatósági ellenőrzése
  - hiba esetén tartalék variáns (soros) végrehajtása

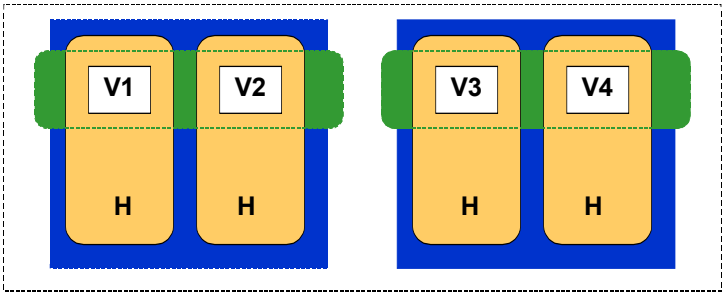


Összehasonlítás

Tulajdonság/típus	**N-verziós programozás**	**Javító blokkok**
Ellenőrzés	Szavazás, relatív	Elfogadhatóság, abszolút
Végrehajtás	Párhuzamos	Soros
Időigény	Leglassabb variáns (vagy time-out)	Hibák számától függő
Redundancia	Mindig	Csak hiba esetén
Tolerált hibák	[N/2-1]	N-1
Hibakezelés	Maszkolás	Helyreállítás

Példa: Airbus A-320

- Páronként önellenőrző végrehajtás (4 variáns: V1, V2, V3, V4 külön-külön hardveren)
- "Aktuális pár" működik és átkapcsolás hiba esetén
- Állandósult hardver hiba: Ismételten hibás pár lekapcsol



8.3. Időleges hardver hibák kezelése

Szoftver redundancia

- ismételt végrehajtás esetén a hiba nem jelentkezik
- hibahatások kiküszöbölése a fontos

A hiba kezelhető hibamentes állapot beállításával (és ismételt végrehajtással)

Feladatok:

a.) Hibadetektálás
b.) Hibahatás felmérése
c.) Helyreállítás
d.) Meghibásodás (hibaok) kezelése

a.) Hibadetektálás:

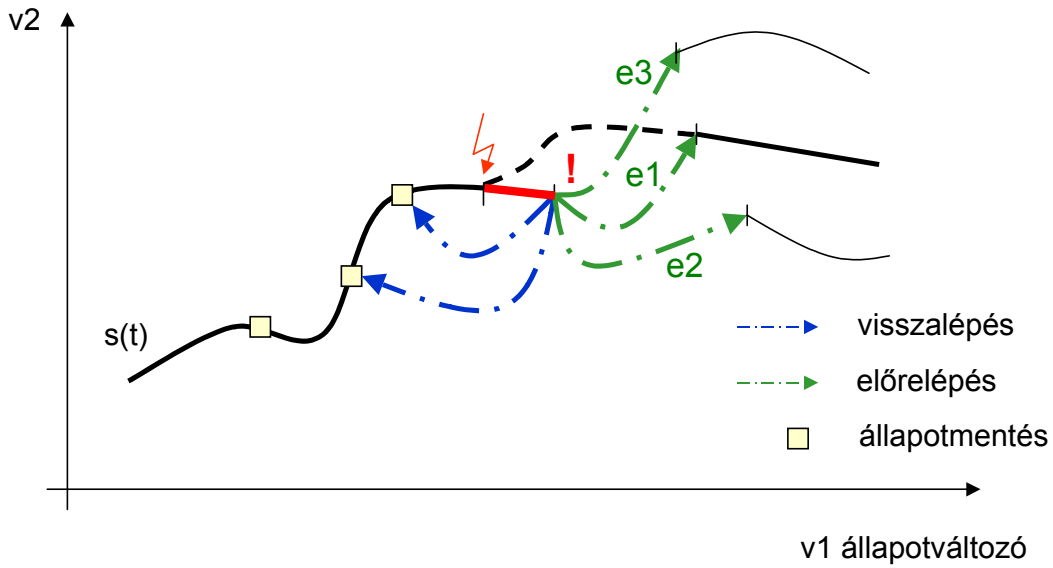
- Alkalmazásfüggetlen: operációs rendszer szintje
  - illegális utasítások felismerése
  - védelmi szintek, jogosultságok (memóriahozzáférés) vizsgálata
- Alkalmazásfüggő módszerek:
  - időzítés ellenőrzése
  - visszahelyettesítés (algoritmus)
  - hihetőségvizsgálat
  - struktúra ellenőrzés
  - diagnosztikai ellenőrzés

b.) Hibahatások felmérése

- Hibadetektálás késleltetési ideje alatt a hiba terjed (pl. processzek, processzorok között)
- Hibaterjedés behatárolása: interakciók követése
  - atomi jellegű műveletek kialakítása
  - bemeneti ellenőrzés
  - interakciók naplózása
- Érintett processzek meghatározása
  - interakciók a hibadetektálás késleltetési ideje alatt

c.) Helyreállítás

- Előrelépő helyreállítás:
  - hibamentes állapot beállítása: szelektív korrekció
  - a detektált hiba és a hibahatás függvénye
  - előre figyelembe vett hibák esetén használható
- Visszalépő helyreállítás:
  - előző hibamentes állapot beállítása
  - hibától függetlenül megvalósítható
  - állapotmentés és visszaállítás lehetősége szükséges feltétel minden komponensre
- Kompenzáció:
  - többlet információ alapján a hibahatás kompenzálható (aktív redundancia)
- Ábrázolás a rendszer állapotterében:



Visszalépő helyreállítás módszerei:

- Állapotmentés alapján
  - checkpoint: állapotmentés (időpontja)
  - műveletek:
    - állapotmentés: időközönként, üzenetek után; stabil tárba
    - visszaállítás: stabil tárból az operatív memóriába
    - eldobás: adott számú checkpoint megtartása
  - pl: “autosave”
- Műveletek visszavonása alapján
  - audit trail: műveletek naplózása
  - pl. többszintű “undo”
- Kombinált módszer (nagyobb időközönként állapotmentés, közben műveletek naplózása)

d.) Meghibásodás (hibaok) kezelése

- Lokalizálás: Tesztelés (diagnosztikai)
- Időleges hibák: előre- vagy visszalépő helyreállítás elég
- Állandósult hibák: helyreállítás nem lesz sikeres (újra hibadetektálás)
  - újrakonfigurálás: hibás komponens feladatainak átvétele degradált működés (biztonsági funkciók)
  - javítás, csere

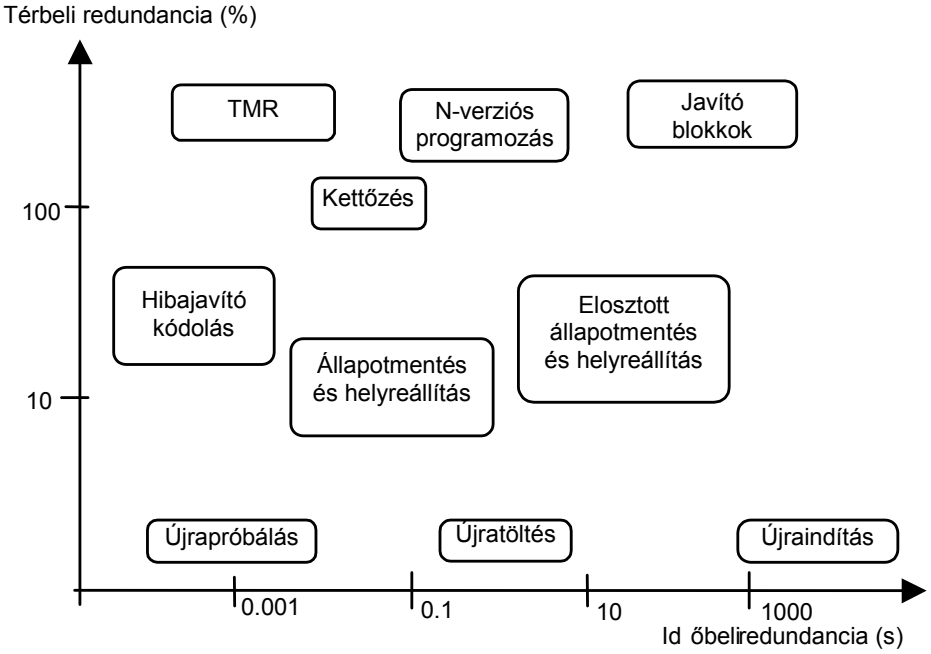
Információ redundancia

- Hibajavító kódolás
  - memóriák. háttértárak, adatátvitel védelme
  - pl. Hamming-kód, Reed-Solomon kódok
- Korlátozott hibajavító képesség
  - hosszú idejű adatstabilitás rossz lehet (“felgyűlnek” a hibák)
  - háttértárak: “memory scrubbing” technika: folyamatos olvasás és javítva visszaírás

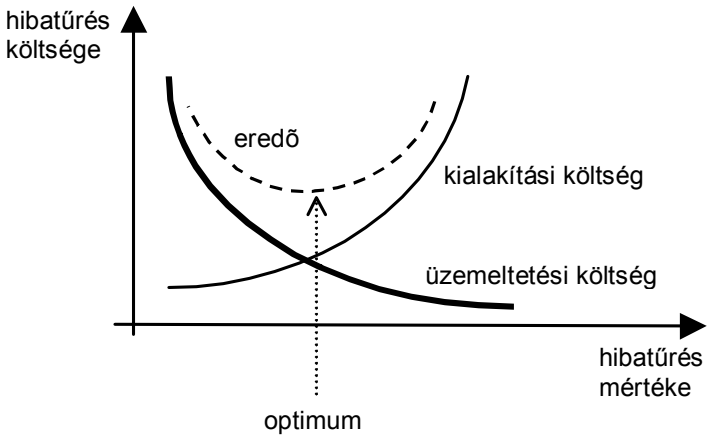
Idő redundancia

- Tiszta eset: utasítás újrapróbálás (retry)
  - alacsony hardver szinten: processzor utasítás
  - időleges hibák esetén hatásos
- Idő redundancia “velejárója” a többi típusnak
  Hard RT rendszerek: tervezési szempont, mennyire garantálható a hibakezelés ideje
  - állandósult hardver hibák: maszkolás, meleg tartalék
  - időleges hardver hibák: előrelépő helyreállítás
  - szoftver tervezési hibák: N-verziós programozás

Időbeli optimalizálás



Költségoptimalizálás



Hibatűrés tesztelése

- Meghibásodás (hiba) előidézése: Hibainjektálás
  - szoftver:
    - hiba létrehozása (rendszerállapot megváltoztatása)
      regiszterek, memóriabitek átállítása (pl. Unix ptrace())
    - rugalmas, gyors
    - hibaokoknak való megfelelés kérdéses
  - hardver:
    - meghibásodás (hibaok) létrehozása:
      busz jelek kényszerítése, tápfeszültség tüske, nehézion-besugárzás
    - hardverfüggő, lassú
  - hibrid
- Hatás monitorozása (működés közben)