15 VALÓS IDEJŰ SZOFTVEREK TERVEZÉSE

TÉMAKÖRÖK

A fejezet célja a valós idejű rendszerek tervezésében használható technikáknak, valamint néhány tipikus valós idejű rendszer architektúrájának bemutatása. Mire a fejezet végére érünk:

- meg fogjuk ismerni a valós idejű rendszerek fogalmait, és meg fogjuk tudni, hogy általában miért konkurens folyamatok segítségével valósítják meg őket;
- meg fogunk ismerkedni a valós idejű rendszerek számára kialakított tervezési folyamattal;
- meg fogjuk érteni a valós idejű operációs rendszerek szerepét;
- meg fogjuk érteni a figyelő- és vezérlő-, valamint az adatgyűjtő rendszerek általános folyamatarchitektúráját.

TARTALOM

Rengeteg rendszer vezérlését végzik számítógéppel, az egyszerű háztartási gépektől egészen a teljes gyártóberendezésekig. Ezek a számítógépek közvetlenül a hardvereszközökkel lépnek kapcsolatba. Az ilyen rendszerek szoftvere olyan *beágyazott valós idejű szoftver,* amelynek reagálnia kell a hardver által kiváltott eseményekre, és ezekre válaszul vezérlőjeleket bocsát ki. Ez a szoftver egy nagyobb hardverrendszerbe van beágyazva, és a rendszer környezetéből érkező eseményekre valós időben kell válaszolnia.

A valós idejű rendszerek különböznek a szoftverrendszerek egyéb fajtáitól. Helyes működésük attól függ, hogy a rendszer képes-e az eseményekre rövid időn belül válaszolni. A valós idejű rendszerek definíciója a következő:

A valós idejű rendszer olyan szoftverrendszer, amelyben a rendszer helyes működése egyrészt a rendszer által adott eredményektől, másrészt attól az időtől függ, amennyi idő alatt ez az eredmény előáll. Gyengén valós idejű a rendszer, ha az korlátozottan

működik, ha a meghatározott időn belül az eredmények nem állnak elő. Az erősen valós idejű rendszer olyan rendszer, amelynek műveletei érvénytelenek, ha az eredmények nem állnak elő a meghatározott időn belül.

Az időben történő válaszadás minden beágyazott rendszer esetén fontos tényező, azonban bizonyos esetekben nincs szükség nagyon gyors válaszra. Például a könyv több fejezetében is példaként használt inzulinadagoló rendszer is beágyazott rendszer, azonban amíg rendszeres időközönként ellenőriznie kell a vércukorszintet, külső eseményekre nem kell nagyon gyorsan válaszolnia. Éppen ezért ebben a fejezetben más példával illusztrálom a valós idejű rendszerek tervezésének alapjait.

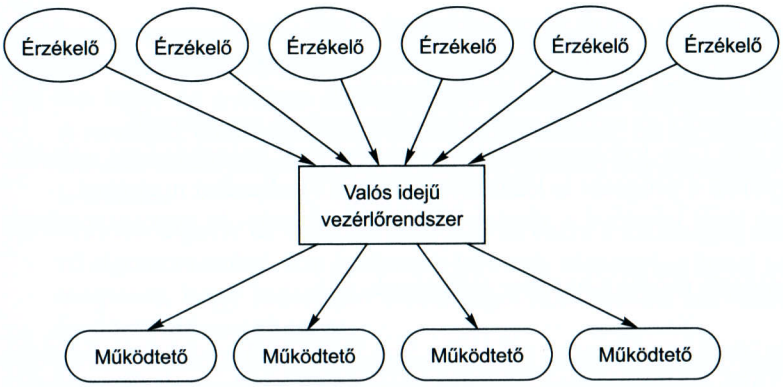
A valós idejű rendszereket egyfelől inger/válasz rendszereknek tekinthetjük. Egy adott ingerre a rendszernek megfelelő választ kell adnia. A valós idejű rendszer viselkedése éppen ezért a rendszert érő ingerek, a rájuk adott válaszok és a válaszidők felsorolásával adható meg. Az ingerek két csoportba oszthatók:

1. *Periodikus ingerek.* A periodikus ingerek meghatározott időközönként fordulnak elő. Például a rendszer 50 milliszekundumonként megvizsgálhat egy érzékelőt, és annak értékétől (inger) függően valamilyen tevékenységet hajthat végre (válasz).
2. *Aperiodikus ingerek.* Az aperiodikus ingerek rendszertelenül keletkeznek. Ezeket általában a számítógép megszakítási mechanizmusa kezeli. Ilyen inger például egy olyan megszakítás, amely azt jelzi, hogy az I/O-művelet befejeződött és az adatok elérhetők a pufferben.

A periodikus ingereket valós idejű rendszerekben általában a rendszerrel kapcsolatban lévő érzékelők váltják ki. Ezek a rendszer környezetének állapotáról adnak információt. A válaszokat a hardverelemeket vezérlő működtetőkhöz irányítják, amelyek majd befolyásolják a rendszer környezetét. Az aperiodikus ingereket akár a működtetők, akár az érzékelők is kiválthatják. Ezek gyakran valamilyen rendkívüli körülményt jelölnek, mint amilyen például egy hardverhiba, amelyet a rendszernek kezelnie kell. A beágyazott valós idejű rendszereknek ez az érzékelő-rendszer-működtető modellje látható a 15.1. ábrán.

Egy valós idejű rendszernek tudnia kell válaszolnia a különböző időben előforduló ingerekre. Éppen ezért architektúráját úgy kell megadni, hogy a vezérlés az inger beérkezését követően a lehető legrövidebb idő alatt eljusson a megfelelő kezelőhöz. Ez szekvenciális programokban kivitelezhetetlen. A valós idejű rendszereket éppen ezért konkurens, együttműködő folyamatokként szokás tervezni. Ezen folyamatokat kezelése céljából a legtöbb valós idejű rendszer futtató rendszere tartalmaz egy *valós idejű operációs rendszert.* Ezen operációs rendszer lehetőségeit a használt valós idejű programozási nyelv futási idejű rendszerén keresztül lehet elérni.

A valós idejű rendszerek inger/válasz modelljének általános volta elvezet egy háromféle folyamatot tartalmazó általános architekturális modellhez. Minden érzékelőtípus rendelkezik érzékelőkezelő folyamattal; a számítási folyamatok a



15.1. ábra. Valós idejű rendszer általános modellje

rendszer által megkapott ingerekre adandó válasz kiszámítását végzik; a működtető-vezérlő folyamatok pedig a működtető műveleteit irányítják. Ez a modell lehetővé teszi az adatok érzékelőktől történő gyors begyűjtését (mielőtt még a következő bemenő adat elérhetővé válna), és megengedi, hogy a feldolgozás, valamint a velük kapcsolatban lévő működtetők válasza csak később legyen végrehajtva.

Ebből az általános architektúrából számos olyan különböző alkalmazásarchitektúra származtatható, amelyek kibővítik a 13. fejezetben tárgyalt architektúrákat. A valós idejű alkalmazásarchitektúrák az eseményvezérelt architektúra példányai, ahol az események az ingerek – közvetlen vagy közvetett – hatására következnek be. Ebben a fejezetben két újabb alkalmazásarchitektúrát mutatok be, a figyelő- és vezérlőrendszerek (15.3. alfejezet), valamint az adatgyűjtő rendszerek (15.4. alfejezet) architektúráit.

A valós idejű rendszerek fejlesztésére használt programozási nyelveknek lehetőséget kell biztosítaniuk a hardver elérésére, és megjósolhatónak kell lennie, hogy bizonyos műveletek mennyi idő alatt végezhetők el. Az erősen valós idejű rendszereket néha még ma is assembly nyelven programozzák, így a szoros határidők tarthatók. A hatékony kódot előállítani képes rendszerszintű nyelveket – mint amilyen például a C is – szintén széles körben használják.

Egy C-hez hasonlóan alacsony szintű programozási nyelv használatának előnye, hogy nagyon hatékony programokat lehet benne fejleszteni. Ezek a nyelvek azonban nem tartalmaznak a konkurencia vagy az osztott erőforrások kezelésére szolgáló konstrukciókat. Ezeket a valós idejű operációs rendszer rendszerhívásainak segítségével valósítják meg, ezeket azonban a fordító nem tudja ellenőrizni, így valószínűbbé válik egy esetleges programozási hiba. Ezeket a programokat ráadásul gyakran még nehezebb megérteni, mert a valós idejű jellemzők nem explicit módon jelennek meg bennük.

Az elmúlt pár évben nagyszabású munka folyt annak érdekében, hogy a Java nyelvet kibővítsék a valós idejű rendszerek fejlesztéséhez szükséges eszközökkel (Nilsen, 1998; Higuera-Toledano és Issarny, 2000; Hardin *és mások*, 2002). Ez a munka a nyelvnek a következő alapvető valós idejű problémák megoldása céljából történő változtatását foglalja magában:

1. Nem lehet megadni a szálak végrehajtásának idejét.
2. A szemétgyűjtő mechanizmus ellenőrizhetetlen, bármelyik pillanatban elkezdődhet. Ezért a szálak időzítése kiszámíthatatlan.
3. Nem lehet megtudni az osztott erőforrásokhoz tartozó sor méretét.
4. A Java Virtuális Gép implementációja számítógépről számítógépre változó, így akár ugyanaz a program is különböző időbeli viselkedést mutathat.
5. A nyelv nem teszi lehetővé a részletes futási idejű hely- és processzorelemzést.
6. Nincs szabványos módja a hardver elérésének.

A Java valós idejű verziói, mint például a Sun J2ME-je (Java 2 Micro Edition) már elérhető. Több szoftvergyártó is szállít valós idejű rendszerek fejlesztésére átdolgozott JVM-implementációt. Ezek a fejlesztések azt jelentik, hogy a Java nyelvet egyre inkább fogják valós idejű programozási nyelvként is használni.

15.1. RENDSZERTERVEZÉS

Mint ahogyan azt a 2. fejezetben már fejtegettük, a rendszertervezés folyamatának része annak eldöntése is, hogy a rendszer mely képességeit implementáljuk szoftveresen, és melyeket valósítjuk meg hardveres úton. A fogyasztási cikkekbe beágyazott valós idejű rendszerek – ilyen például egy mobiltelefon rendszere – esetén a hardver költsége és energiafogyasztása kritikus. A beágyazott rendszerek támogatására tervezett speciális processzorok használhatók, illetve, bizonyos rendszerek esetén speciális célhardverek tervezésére és megépítésére lehet szükség.

Ez azt jelenti, hogy a fentről lefelé haladó tervezési stratégia – ahol a tervezés egy absztrakt modellel indul, amit aztán fel kell bontani, és lépésenként kibontani – használata a legtöbb valós idejű rendszer esetén nem célszerű. A hardverekről szóló alacsony szintű döntésekkel, a szoftvertámogatásról és a rendszer időzítéséről a folyamat korai szakaszában kell dönteni. Ezek korlátozzák a rendszertervezők lehetőségeit, és esetleg azt is jelenthetik, hogy olyan további szoftveres funkcionalitásra lesz szükség, mint amilyen az energiakezelés.

A valós idejű szoftverek tervezési folyamatának középpontjában nem az objektumok és a funkciók, hanem az események (azaz az ingerek) állnak. A tervezésnek számos átfedő lépése van:

1. Meg kell határozni a rendszer által feldolgozandó ingereket és az azokra adandó választ.
2. Minden ingerre és az ahhoz kapcsolódó válaszra meg kell határozni az inger és a hozzá tartozó válasz feldolgozására vonatkozó időzítési megszorításokat.
3. Választani kell a rendszer számára egy végrehajtási platformot, vagyis meg kell határozni az alkalmazandó hardvert és valós idejű operációs rendszert. A választást befolyásoló tényezők között szerepelnek a rendszer időre vonat-

kozó megszorításai, a rendelkezésre álló energia korlátozásai, a fejlesztőcsapat tapasztalata és a leszállított rendszer tervezett ára.

4. Az inger és a válasz feldolgozását konkurens folyamatokba kell gyűjteni. A rendszer architektúrájára jó általános modell, ha az ingerek és a válaszok minden osztályához egy folyamatot rendelünk, mint ahogyan az a 15.2. ábrán is látható.
5. Minden ingerre és válaszra meg kell tervezni a szükséges számításokat végző algoritmusokat. Ezt gyakran a tervezés viszonylag korai szakaszában kell megtenni, hogy kiderüljön a szükséges feldolgozási műveletek és az azokra fordított idő mennyisége.
6. Ki kell alakítani az ütemezési rendszert, amely biztosítja, hogy a folyamatok időben elinduljanak és határidőre befejeződjenek.

15.2. ábra. Érzékelő/működtető vezérlőfolyamatok

Ezen tevékenységek sorrendje függ a kifejlesztendő rendszer fajtájától, folyamat- és platformkövetelményeitől. Bizonyos esetekben egy meglehetősen absztrakt megközelítést követve kiindulhatunk az ingerekből és a hozzájuk kapcsolódó feldolgozásból, és a végrehajtási platformra vonatkozó döntést elég a folyamat késői szakaszában meghozni. Más esetekben a hardver és az operációs rendszer kiválasztása megelőzi a szoftvertervezés kezdetét, és a tervet a hardver képességeinek figyelembevételével kell elkészíteni.

A valós idejű rendszerek folyamatait koordinálni kell. A folyamatkoordinációs mechanizmusok biztosítják a kölcsönös kizárást a megosztott erőforrásokhoz. Ha egy folyamat éppen módosít egy megosztott erőforrást, más folyamatoknak nem szabad azt megváltoztatniuk. A kölcsönös kizárást biztosító módszerek között meg kell említeni a szemaforokat (Dijkstra, 1968), a monitorokat (Hoare, 1974) és a kritikus régiókat (Brinch-Hansen, 1973). Ezeket a mechanizmusokat a legtöbb, operációs rendszerekről szóló könyv részletesen tárgyalja (Tanenbaum, 2001; Silberschatz *és mások*, 2002).

Ha kiválasztottuk a végrehajtási platformot, megterveztük a folyamatarchitektúrát és meghatároztuk az ütemezési politikát, le kell ellenőriznünk, hogy a rendszer megfelel-e az időzítési követelményeknek. Ezt az egyes komponensek időbeli viselkedésének ismeretében statikus elemzéssel vagy szimuláció segítségével végezhetjük el. Ez az elemzés adott esetben azt is feltárhatja, hogy a rendszer nem megfelelően teljesít. Ekkor a folyamatarchitektúrát, az ütemezési politikát, a végrehajtási platformot, vagy esetleg ezek mindegyikét újra kell tervezni a rendszer teljesítményének növelése érdekében.

Egy valós idejű rendszer időzítésének elemzése bonyolult. Az aperiodikus ingerek megjósolhatatlan természete miatt a tervezőknek bizonyos feltételezéssel kell élniük ezen ingerek egységnyi idő alatti bekövetkezésének (azaz a szolgáltatáskérésnek) a valószínűségére vonatkozóan. Azonban nem biztos, hogy ezek a feltételezések helyesek, így elképzelhető, hogy az átadást követően a rendszer teljesítménye nem megfelelő. A valós idejű rendszerek teljesítményelemzési módszereit Cooling könyve tárgyalja (Cooling, 2003).

Mivel a valós idejű rendszereknek meg kell felelniük időzítési megszorításaiknak, erősen valós idejű rendszerek esetén nem célszerű olyan tervezési stratégiákat alkalmazni, amelyek implementációs többletterhelést okoznak. Például az objektumorientált fejlesztés elrejti az adatreprezentációt, és az attribútumértékekhez az objektum műveleteinek segítségével lehet hozzáférni. Ez az objektumorientált rendszerek esetén jelentős teljesítménybeli többletterhelést jelent, mert az attribútumok eléréséhez többletkód végrehajtására van szükség, és az ebből eredő teljesítménycsökkenés ellehetetlenítheti a határidők betartását.

Az időre vonatkozó megszorítások és egyéb követelmények azt is jelenthetik, hogy a rendszer néhány funkcióját – például a jelfeldolgozást – külön e célra tervezett hardverrel kell megvalósítani. A hardverelemekbe történő beépítéssel sokkal jobb teljesítményt lehet elérni, mint a neki megfelelő szoftverrel. Meg lehet határozni a rendszer szűk feldolgozási keresztmetszeteit, és azokat hardverrel helyettesítve elkerülhető a szoftverek költséges optimalizációja.

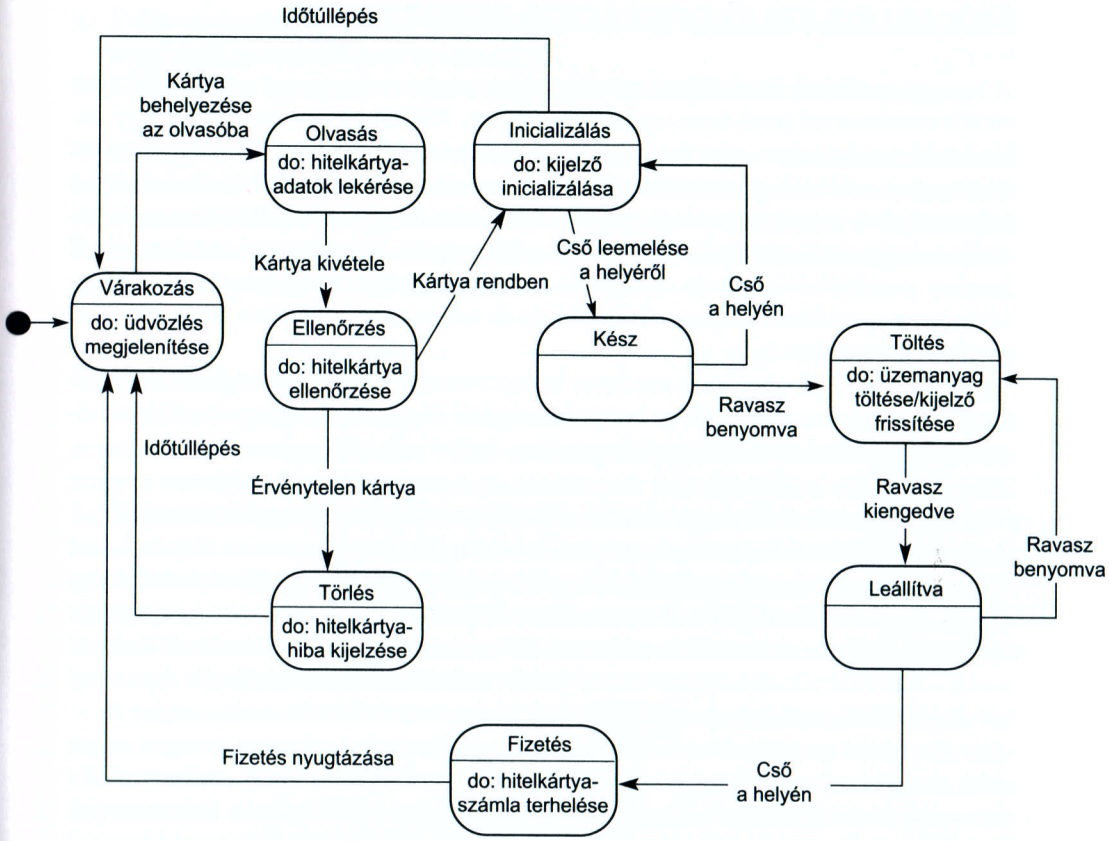
15.1.1. Valós idejű rendszerek modellezése

A valós idejű rendszereknek a rendszertelen időközönként bekövetkező eseményekre kell válaszolniuk. Ezek az események (vagy ingerek) gyakran azt eredményezik, hogy a rendszer más állapotba kerül. Éppen ezért a valós idejű rendszerek leírásának egyik módja a 8. fejezetben bemutatott állapotgép-modellezés.

Az állapotgépmodellek segítségével nyelvfüggetlen módon adható meg a valós idejű rendszerek terve, éppen ezért szerves részét képezik a valós idejű tervezési módszereknek (Gomaa, 1993). Az UML az állapotmodellek fejlesztését az állapotdiagramok segítségével támogatja (Harel, 1987; 1988). Az állapotdiagramok strukturálják az állapotmodelleket, így állapotok egy csoportját különálló entitásként tekinthetjünk. Az UML valós idejű rendszerek fejlesztésében betöltött szerepét Douglass tárgyalja (Douglass, 1999).

A rendszer állapotmodellje feltételezi, hogy a rendszer minden időben számos lehetséges állapot közül pontosan egyben van. Egy inger egy más állapotba történő átmenetet okozhat. Például ha egy szelepet vezérlő rendszer a „Szelep nyitva" állapotban kezelői utasítást (ingert) kap, akkor átkerülhet a „Szelep zárva" állapotba.

Ezt a rendszer-modellezési megközelítést a 8. fejezetben már egy egyszerű mikrohullámú sütő modelljén keresztül bemutattam. A 15.3. ábrán egy másik állapotgépmodell látható, amely egy benzinkútba beágyazott üzemanyag-adagoló szoftver működését ábrázolja. A lekerekített téglalapok a rendszer állapotait



15.3. ábra. Egy benzinkút állapotgépmodellje

jelzik, míg a címkézett nyilak azokat az ingereket reprezentálják, amelyek egyik állapotból a másikba kényszerítik a rendszert. Az állapotdiagramban használt nevek beszédesek, és a hozzájuk kapcsolt információk a működtetők által végzett tevékenységeket vagy a megjelenített információkat jelölik.

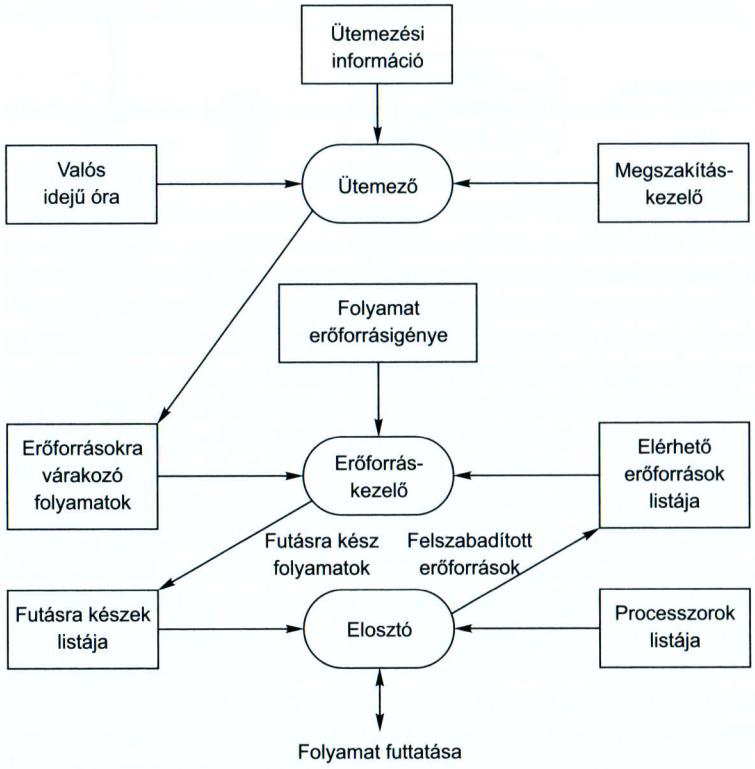
Az üzemanyag-adagoló rendszert úgy tervezték, hogy felügyelet nélkül is lehessen működtetni. A vásárló behelyezi a hitelkártyáját a kútba épített kártyaolvasóba. Ez az **Olvasás** állapotba történő átmenetet eredményezi, amikor is a kártya adatai leolvasásra kerülnek, és megkérjük a vásárlót, hogy vegye ki a kártyáját. A rendszer ekkor az **Ellenőrzés** állapotba kerül, ekkor történik meg a kártya ellenőrzése. Ha a kártya érvényes, a rendszer inicializálja a kutat, és ha a vásárló az üzemanyagcsövet kiveszi a helyéről, az működésre kész. A cső végén lévő ravasz meghúzása az üzemanyag töltését okozza, ami a ravasz felengedésekor áll meg (az egyszerűség kedvéért figyelmen kívül hagytam az üzemanyag kiömlését gátló nyomásérzékelőt). Miután a vásárló a csövet visszahelyezi a helyére, a rendszer a **Fizetés** állapotba kerül, amikor a számlájára ráterhelődik a tankolás összege.

15.2. VALÓS IDEJŰ OPERÁCIÓS RENDSZEREK

A legegyszerűbbek kivételével minden beágyazott rendszer egy valós idejű operációs rendszerrel (real-time operating system, RTOS) együtt működik. Egy valós idejű rendszerben egy valós idejű operációs rendszer kezeli a folyamatokat és végzi az erőforrás-kiosztást; az ingerek kezelése céljából elindítja és leállítja a folyamatokat; végzi a tár- és processzorkiosztást. Nagyon sok RTOS-termék létezik, a fogyasztói eszközök számára készült nagyon kis, egyszerű rendszerektől kezdve a mobiltelefonok és -eszközök számára készült bonyolult rendszerekig, ideértve a speciálisan folyamatirányításra és telekommunikációra tervezett operációs rendszereket is.

Egy valós idejű operációs rendszer komponensei (15.4. ábra) a fejlesztendő valós idejű rendszer méretétől és bonyolultságától függnek. A legegyszerűbb rendszerek kivételével általában tartalmaznak:

1. *Valós idejű órát:* a folyamatok periodikus ütemezéséhez biztosít információt.
2. *Megszakításkezelőt:* kezeli az aperiodikus kérelmeket.
3. *Ütemezőt:* ez a komponens felelős azért, hogy a futtatható folyamatokat megvizsgálja, és közülük kiválasszon egyet futásra.



15.4. ábra. Valós idejű operációs rendszer komponensei

4. *Erőforrás-kezelőt:* a futásra kész folyamatnak az erőforrás-kezelő osztja ki a megfelelő memóriát és processzoridőt.
5. *Elosztót:* ez a komponens felel a folyamat végrehajtásának elindításáért.

A nagy – például folyamatirányító, telekommunikációs – rendszerek számára készült valós idejű operációs rendszerek további lehetőségeket is tartalmazhatnak: lemezkezelőt; olyan hibakezelő eszközöket, amelyek felismerik és jelentik a rendszerhibákat; konfigurációkezelőt, amely a valós idejű alkalmazások dinamikus újrakonfigurálását teszi lehetővé.

15.2.1. Folyamatkezelés

A valós idejű rendszereknek a külső eseményeket gyorsan kezelniük kell, és bizonyos esetekben be kell tartaniuk az események feldolgozására előírt határidőt. Ez azt jelenti, hogy az eseménykezelő folyamatokat úgy kell ütemezni, hogy időben észleljék az eseményt, és megfelelő mennyiségű processzor-erőforrásnak kell rendelkezésre állnia annak érdekében, hogy a határidőt tartani lehessen. A valós idejű operációs rendszerben a folyamatkezelő felel a folyamatok végrehajtásra történő kiválasztásáért, a processzor- és memória-erőforrások kiosztásáért, a folyamatok elindításáért, leállításáért.

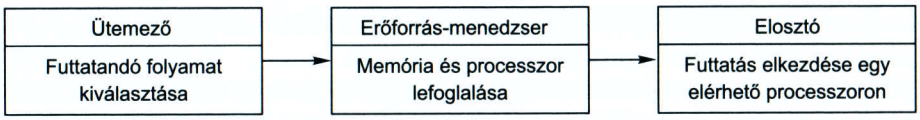
A folyamatkezelő különböző prioritással bíró folyamatok kezelését végzi. Az olyan ingerek számára, amelyek egyes rendkívüli eseményekhez kötődnek, nélkülözhetetlen, hogy a megadott időhatáron belül feldolgozásra kerüljenek. Más folyamatokat esetleg lehet biztonságosan késleltetni, ha egy kritikusabb folyamat igényel kiszolgálást. Ebből következik, hogy egy valós idejű operációs rendszernek a rendszer folyamatainak legalább két prioritási szintjét kell tudni kezelnie:

1. *Megszakításszintű.* Ez a legmagasabb prioritási szint. Olyan folyamatok kapják, amelyeknek nagyon gyors válaszra van szüksége. Az egyik ilyen folyamat a valós idejű óráé.
2. *Órajelszintű.* Ilyen szintű prioritást kapnak a periodikus folyamatok.

Egy újabb prioritási szintet adhatunk azoknak a háttérben futó folyamatoknak (mint amilyen például egy önellenőrző folyamat), amelyeknek nem kell eleget tenniük a valós idejű határidőknek. Ezek a folyamatok akkor futnak, ha van szabad processzorkapacitás.

Minden prioritási szinten belül, a különböző osztályokba tartozó folyamatokhoz különböző prioritások rendelhetők. Például több megszakításcsatorna is létezhet. Nagyon gyors eszköz megszakításának feldolgozása megelőzheti egy lassabbét, nehogy információvesztés következzen be. A folyamatokhoz prioritások oly módon történő hozzárendelése, hogy azok időben kiszolgálásra kerüljenek, általában kiterjedt elemzést és szimulációt igényel.

A periodikus folyamatokat megadott időközönként végre kell hajtani, hogy azok adatgyűjtést végezzenek és vezéreljék a működtetőt. A legtöbb valós idejű



15.5. ábra. Egy folyamat elindításához szükséges RTOS-műveletek

rendszerben a periodikus folyamatoknak számos csoportja van. Ezek különböző periódussal (a folyamat futtatásai közötti idő), futási idővel és határidőkkel (az az idő, amikorra a feldolgozásnak be kell fejeződnie) rendelkeznek. Az alkalmazás időzítési követelményei segítségével az operációs rendszer úgy szervezi a periodikus folyamatok végrehajtását, hogy mindannyian tartani tudják határidejüket.

A 15.5. ábrán a periodikus folyamatkezelés esetén az operációs rendszer által végrehajtandó tevékenységek láthatók. Az ütemező megvizsgálja a periodikus folyamatokat, majd kiválaszt közülük egyet futásra. Ez a választás függ a folyamat prioritásától, periódusától, várható futási idejétől és a kész folyamatok határidőitől. Néha két, különböző határidővel rendelkező folyamatot ugyanazon órajel hatására kellene végrehajtani. Ilyen esetben az egyiket el kell halasztani, feltéve, ha határideje még tartható.

Az aszinkron eseményekre válaszolni köteles folyamatok általában megszakításvezéreltek. A számítógép megszakítási mechanizmusa a vezérlést egy előre megadott memóriacímre adja, amely egy, a megszakítást kiszolgáló egyszerű és gyors rutinra ugró utasítást tartalmaz. A kiszolgáló rutinnak le kell tiltania a további megszakításokat, hogy elkerülje saját maga megszakítását. Fel kell ismernie a megszakítás okát, és nagy prioritással el kell indítania egy, a megszakítást okozó ingert kezelő folyamatot. Néhány nagy sebességű adatgyűjtő rendszerben a megszakításkezelő a megszakítás bekövetkezésekor elérhető adatokat egy pufferbe menti a későbbi feldolgozás céljából. A megszakítások ezután újra engedélyezhetők, a vezérlés pedig visszakerül az operációs rendszerhez.

Minden időpillanatban számos, különböző prioritással rendelkező folyamat várhat futásra. Végrehajtásuk sorrendjéről a rendszer-ütemezési politikát megvalósító folyamatütemező dönt. Két alapvető ütemezési stratégia ismeretes:

1. *Nem preemptív ütemezés.* Egy futásra kiválasztott folyamat addig fut, amíg el nem érjük a végét vagy amíg valamilyen okból kifolyólag blokkolttá nem válik (például bemenetre vár). Ez eltérő prioritású folyamatok esetén problémákat okoz, mert a nagyobb prioritással rendelkező folyamatnak meg kell várnia a kisebb prioritású befejeződését.
2. *Preemptív ütemezés.* Egy futó folyamat végrehajtása leállítható, ha egy nagyobb prioritással rendelkező folyamat igényel kiszolgálást. A nagyobb prioritású folyamatnak elsőbbsége van az alacsonyabb prioritásúval szemben, így ő jut a processzorhoz.

Különféle, az említett két stratégiába illő ütemezési algoritmusokat fejlesztettek már ki, többek között a ciklikus időszeleteléses (ahol a folyamatok felváltva

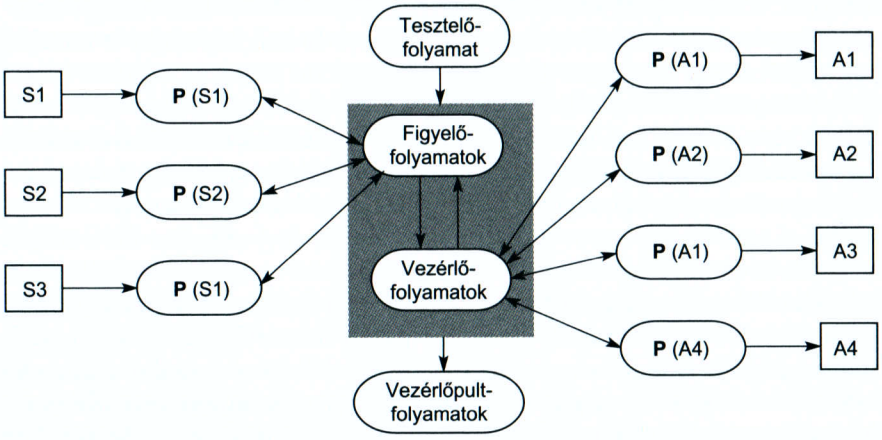
kerülnek végrehajtásra), a monoton gyakoriságon alapuló – ahol a legrövidebb periódusú folyamat kap prioritást –, valamint a legrövidebb határidejű folyamatot elsőként futtató ütemezést (Burns és Wellings, 2001).

A végrehajtandó folyamatokra vonatkozó információk az erőforrás-kezelőhöz kerülnek, ez oszt memóriát és – többprocesszoros rendszer esetén – processzort a folyamatnak. A folyamat ezután egy készenléti listára kerül, ahol a futásra kész folyamatok helyezkednek el. Mikor a processzor befejezi az éppen futó folyamat végrehajtását és elérhetővé válik, munkába lép az elosztó: a készenléti listán végigfutva kiválaszt egy, az adott processzoron végrehajtható folyamatot, és megkezdi annak végrehajtását.

15.3. FIGYELŐ- ÉS VEZÉRLŐRENDSZEREK

A figyelő- és vezérlőrendszerek a valós idejű rendszerek egyik nagyon fontos csoportját alkotják. Ellenőrzik a rendszer környezetéről információt biztosító érzékelőket és az érzékelt értéktől függően tevékenységeket hajtanak végre. A figyelőrendszerek akkor lépnek akcióba, ha az érzékelő valamilyen nem várt értéket érzékel. A vezérlőrendszerek folyamatosan irányítják a hardveres működtetőket, a velük kapcsolatban lévő érzékelők értékeitől függően.

A 15.6. ábrán a figyelő- és vezérlőrendszerek jellemző architektúrája látható. Minden megfigyelt érzékelőfajta saját figyelőfolyamattal rendelkezik, ahogyan minden vezérelt működtetőfajta is saját vezérlőfolyamattal. A figyelőfolyamat összegyűjti és integrálja az adatokat, mielőtt átadná a vezérlőfolyamatnak, amely ezen adatok alapján dönt, és kiadja a megfelelő vezérlőutasítást. Egyszerű rendszerekben a figyelő- és vezérlőfeladatok egyazon folyamatba is integrálhatók. Az ábrán két másik folyamat is látható, amelyek szintén részei lehetnek a figyelő- és vezérlőrendszereknek. Ezek a hardvertesztelő programokat futtató tesztelőfolya-



15.6. ábra. Egy figyelő- és vezérlőrendszer általános architektúrája

mat és egy vezérlőpult-folyamat, amely a rendszer vezérlőpultját vagy operátor konzolját kezeli.

A figyelő- és vezérlőrendszerek tervezésének bemutatása céljából tekintsük a következő példát, amely egy irodaépületben telepíthető riasztórendszert ír le:

> *Egy kereskedelmi épületbe történő behatolás ellen védő riasztórendszer vezérlését végző szoftvert kell megvalósítani. A rendszer különféle érzékelőket használ. A különálló szobákban mozgásérzékelőket kell felszerelni, a földszinti ablakokra olyan érzékelő kell, ami érzékeli, hogy az ablakot betörték-e, míg a folyosói ajtókat nyitásérzékelővel kell ellátni. A rendszer 50 ablak-, 30 ajtó- és 200 mozgásérzékelőt tartalmaz.*
>
> *Ha egy érzékelő behatolót észlel, a rendszer automatikusan értesíti a helyi rendőrséget, és egy beszédszintetizátor segítségével közli a riasztás helyét. Felkapcsolja az aktív érzékelő közelében lévő szobák világítását és bekapcsolja a riasztóhangot. A riasztórendszer normál körülmények között hálózati feszültségről működik, de fel van szerelve tartalék akkumulátorral is. A hálózati feszültséget külön áramkörfigyelő ellenőrzi, amely feszültségesés esetén megszakítást küld a riasztórendszernek.*

A rendszer egy „gyengén" valós idejű rendszer, szigorú időzítési követelmények nélkül. Az érzékelőknek nem kell nagy sebességű eseményeket észlelniük, ezért elegendő azokat másodpercenként csupán kétszer leolvasni. A példa könnyebb érthetősége kedvéért a tesztelő- és megjelenítő folyamatok elhagyásával leegyszerűsítettem a tervet.

A tervezési folyamat a 15.1. alfejezetben megtárgyalt lépések szerint halad, vagyis első lépésként meg kell határozni azt, hogy a rendszer milyen aperiodikus ingereket kaphat és hogyan kell azokra reagálnia. A korábban leírt egyszerűsítések miatt figyelmen kívül hagyjuk a rendszer önellenőrző eljárásai által generált, valamint a rendszer tesztelése, illetve téves riasztás esetén történő kikapcsolásával kiváltott külső ingereket. Ez azt jelenti, hogy csak két feldolgozandó ingerfajta létezik:

1. *Áramszünet.* Ezt az ingert az áramkörfigyelő váltja ki. Erre válaszul egy elektronikus kapcsolónak küldött jelzés segítségével át kell kapcsolni a tartalék akkumulátorra.
2. *Riasztás behatolás miatt.* Ezt az ingert a rendszer érzékelőinek valamelyike váltja ki. Válaszképpen meg kell határozni az aktív érzékelő szobaszámát, értesíteni kell a rendőrséget, el kell indítani a hívást intéző beszédszintetizátort, valamint be kell kapcsolni a riasztóhangot és a környéken lévő lámpákat is.

A tervezés következő lépésében meg kell határozni az ingerekhez, illetve a rájuk adandó válaszokhoz kapcsolódó időkorlátokat. Ezek az időkorlátok a 15.7. ábrán láthatók. Az időkorlátokat érzékelőfajtákra lebontva, külön-külön kell megadni, még akkor is, ha azok esetleg több érzékelőfajtára vonatkozóan megegyeznek, mint ahogyan az a jelenlegi esetben is van. Ha külön kezeljük őket, akkor a későbbi esetleges módosítások is könnyebbé válnak, és azt is egyszerűbb kiszámolni, hogy a vezérlőfolyamatot hányszor kell lefuttatni másodpercenként.

Inger/válasz	Időzítési követelmény
Áramszünet-megszakítás	A tartalék akkumulátorra történő átállásnak 50 ms alatt meg kell történnie.
Ajtóriasztó	Minden ajtóriasztó állapotát másodpercenként kétszer le kell olvasni.
Ablakriasztó	Minden ablakriasztó állapotát másodpercenként kétszer le kell olvasni.
Mozgásérzékelő	Minden mozgásérzékelő állapotát másodpercenként kétszer le kell olvasni.
Riasztóhang	A riasztóhangot az érzékelő által keltett riasztás után fél másodpercen belül be kell kapcsolni.
Világítás felkapcsolása	A világítást az érzékelő által keltett riasztás után fél másodpercen belül be kell kapcsolni.
Kommunikáció	Az érzékelő által keltett riasztás után 2 másodpercen belül telefonálni kell a rendőrségnek.
Beszédszintetizátor	A szintetizált üzenetnek az érzékelő által keltett riasztás után 4 másodpercen belül rendelkezésre kell állnia.

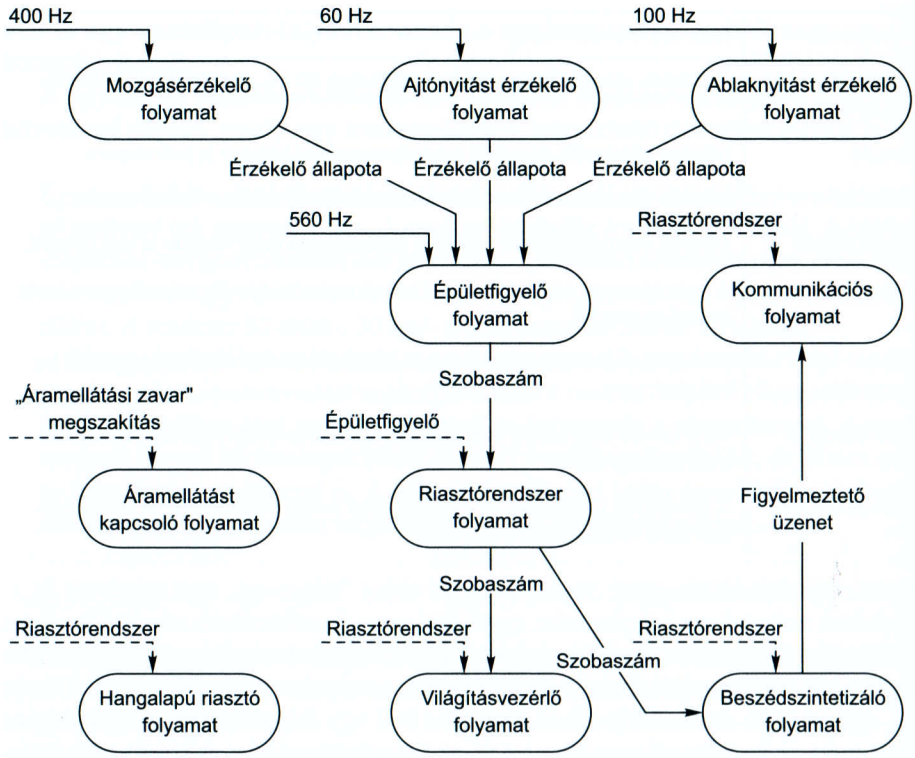
15.7. ábra. Inger/válasz időkorlátok

A következő tervezési lépés a rendszerfunkciók konkurens folyamatokhoz rendelése. A rendszerben háromfajta érzékelő van, amelyeket periodikusan kell leolvasni, így minden érzékelőtípushoz tartoznia kell egy folyamatnak. Megszakításalapú rendszer kezeli az áramszünetet és az áramforrásváltást, a kommunikációs rendszert, a beszédszintetizátort, a hangriasztó rendszert és az érzékelő környezetének világítását kapcsoló kapcsolórendszert. Ezeket a rendszereket egymástól független folyamatok irányítják. Ez a rendszer-architektúra a 15.8. ábrán látható.

A 15.8. ábra jelöléseiben a folyamatokat összekötő címkézett nyilak a folyamatok közötti adatáramlást jelölik, a címke pedig az adatáramlás típusát jelzi. Nem minden folyamat kap adatokat más folyamatoktól. Az áramszünet kezelését végző folyamatnak például nincs szüksége adatokra a rendszer többi részétől.

A folyamatok bal felső részén látható nyilak a vezérlést jelzik. A periodikus folyamatok nyilai folytonos vonallal vannak ábrázolva, a feliratuk pedig az a legkisebb szám, ahányszor a folyamatnak másodpercenként futnia kell. Az aperiodikus folyamatok vezérlő nyilait szaggatott vonallal jelöltük. A nyilak a folyamat ütemezését kiváltó eseménnyel vannak címkézve.

Azt, hogy az egyes folyamatoknak milyen gyakorisággal kell futniuk, az érzékelők száma és a rendszer időkorlátai alapján tudjuk kiszámítani. Például 30 ajtóérzékelő van, amelyek mindegyikét másodpercenként kétszer le kell kérdezni. Ez azt jelenti, hogy a vele kapcsolatban lévő ajtóérzékelő folyamatnak másodpercenként 60-szor kell futnia (60 Hz). Hasonlóan a mozgásérzékelő folyamatnak másodpercenként 400-szor kell futnia, ugyanis 200 mozgásérzékelőre készülünk fel. A működtető folyamatok (azaz a riasztóhang-vezérlő, a világításvezérlő stb.) vezérlőinformációi azt mutatják, hogy ezek a folyamatok a Riasztórendszer vagy az Áramellátási zavar megszakítás folyamattól kapott explicit parancs hatására indulnak el.



15.8. ábra. A riasztórendszer folyamatarchitektúrája

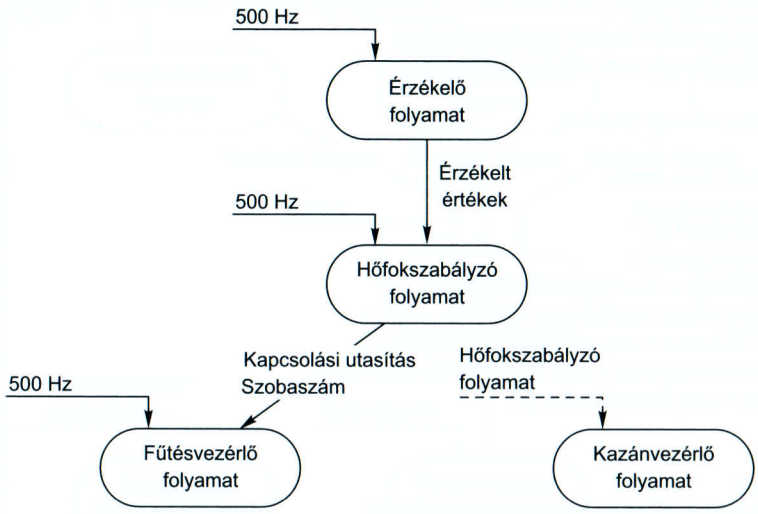
Ezek a folyamatok Javában szálak segítségével implementálhatók. A 15.9. ábrán a BuildingMonitor (Épületfigyelő) folyamat Java-kódja látható, amely a rendszer érzékelőinek leolvasását végzi. Ha ezek behatolót jeleznek, akkor a szoftver aktiválja a riasztórendszert. Ebben a tervben feltettük, hogy a megjegyzésben leírt időkorlátoknak eleget lehet tenni. Ahogy azt már korábban leírtuk, a Java nem rendelkezik olyan lehetőségekkel, amelyek lehetővé tennék a szálak futtatási gyakoriságának megadását.

Miután egy rendszernek kialakult a folyamatarchitektúrája, az ingerfeldolgozást és a válasz generálását végző algoritmusokat kell megtervezni. Mint ahogyan azt már a 15.1. alfejezetben is említettem, ezt a részletes tervezési lépést szükségszerűen a tervezési folyamat elején kell végrehajtani, hogy megbizonyosodjunk arról, hogy a rendszer eleget tud-e tenni a megadott időkorlátoknak. Ha a megfelelő algoritmusok bonyolultak, akkor szükségessé válhat az időkorlátok módosítása. A valós idejű rendszerek algoritmusai általában nagyon egyszerűek, kivéve, ha jelfeldolgozásra van szükség. Csak egy ellenőrzendő memóriacímre, néhány egyszerű számítás elvégzésére vagy egy jel kibocsátására van szükségük. A 15.9. ábrán látható, hogy a riasztórendszerrel is ez a helyzet, ezért az algoritmustervezési szakaszt kihagytam.

A tervezés utolsó lépésében egy ütemező rendszer tervét kell elkészíteni, amely biztosítja a folyamatok olyan ütemezését, hogy teljesíthessék határidői-

```
// A példa teljes Java forrására mutató hiperhivatkozások
// a http://www.software-engin.com/ oldalon találhatók.

class BuildingMonitor extends Thread {

  BuildingSensor win, door, move ;

  Siren   siren = new Siren () ;
  Lights   lights = new Lights () ;
  Synthesizer synthesizer = new Synthesizer () ;
  DoorSensors doors = new DoorSensors (30) ;
  WindowSensors  windows = new WindowSensors (50) ;
  MovementSensors movements = new MovementSensors (200) ;
  PowerMonitor pm = new PowerMonitor () ;

  BuildingMonitor()
  {
    // az érzékelők inicializálása és a folyamatok elindítása
    siren.start () ; lights.start () ;
    synthesizer.start () ; windows.start () ;
    doors.start () ; movements.start () ; pm.start () ;
  }

  public void run ()
  {
    int room = 0 ;
    while (true)
    {
      // a mozgásérzékelők lekérdezése másodpercenként legalább kétszer (400 Hz)
      move = movements.getVal () ;
      // az ablakérzékelők lekérdezése másodpercenként legalább kétszer (100 Hz)
      win = windows.getVal () ;
      // az ajtóérzékelők lekérdezése másodpercenként legalább kétszer (60 Hz)
      door = doors.getVal () ;
      if (move.sensorVal == 1 | door.sensorVal == 1 | win.sensorVal == 1)
        {
          // egy érzékelő behatolást észlelt
          if (move.sensorVal == 1)  room = move.room ;
          if (door.sensorVal == 1)   room = door.room ;
          if (win.sensorVal == 1 )   room = win.room ;

          lights.on (room) ; siren.on () ; synthesizer.on (room) ;
          break ;
        }
    }
    lights.shutdown () ; siren.shutdown () ; synthesizer.shutdown () ;
    windows.shutdown () ; doors.shutdown () ; movements.shutdown () ;

  } // run
} //BuildingMonitor
```

15.9. ábra. Az épületfigyelő folyamat Java-implementációja



15.10. ábra. Egy hőmérséklet-szabályzó rendszer folyamatarchitektúrája

ket. Ebben a példában nincsenek feszes határidők. A folyamatok prioritását úgy kell megadni, hogy minden, érzékelőt leolvasó folyamat ugyanolyan prioritással rendelkezzen. Az áramszünet kezelését végző folyamatnak nagy prioritású, megszakításszintű folyamatnak kell lennie. A riasztórendszert kezelő folyamatok prioritásainak meg kell egyezniük az érzékelő folyamatok prioritásaival.

A behatolás elleni riasztórendszer inkább figyelő-, mintsem vezérlőrendszer, mivel nem rendelkezik az érzékelők értékei által közvetlenül befolyásolt működtetőkkel. Egy épület fűtését irányító rendszer például vezérlőrendszer. Egy ilyen rendszer figyeli a különböző szobák hőmérséklet-érzékelőit és az aktuálisan mért, valamint a szobai hőfokszabályzón beállított hőmérséklettől függően kapcsolja be, illetve le a fűtést. A hőfokszabályzó irányítja a kazán kapcsolását is.

A rendszer folyamatarchitektúrája a 15.10. ábrán látható. Világosan kitűnik, hogy ez hasonlít a behatolás elleni riasztórendszerhez. A példa további kidolgozása az olvasóra van bízva.

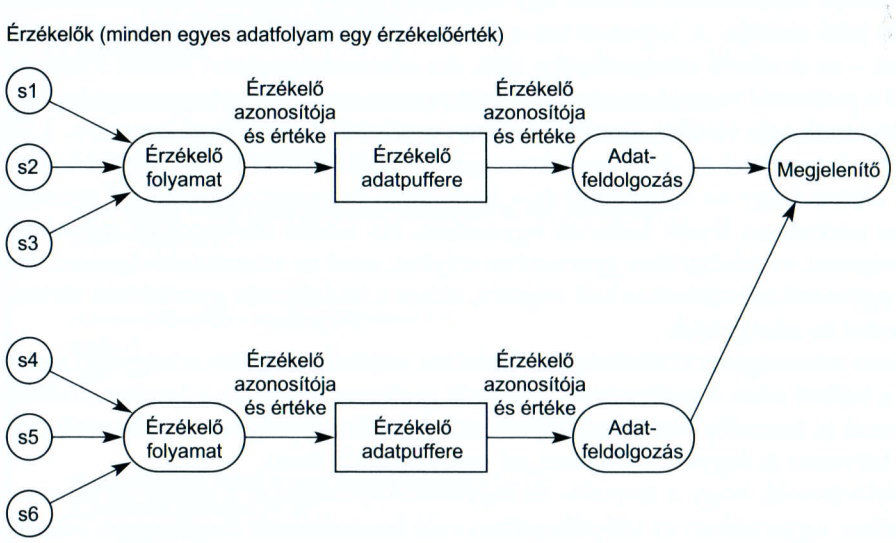
15.4. ADATGYŰJTŐ RENDSZEREK

Az adatgyűjtő rendszerek további feldolgozás és elemzés céljából gyűjtenek adatokat az érzékelőktől. Ilyen rendszereket akkor alkalmazunk, ha az érzékelők által a rendszer környezetéből összegyűjtött rengeteg adat valós idejű feldolgozása nem lehetséges vagy nem szükséges. Az adatgyűjtő rendszereket leginkább tudományos kísérletekben és olyan folyamatirányító rendszerekben használják, ahol valamilyen fizikai folyamat (például egy kémiai reakció) nagyon gyorsan megy végbe.

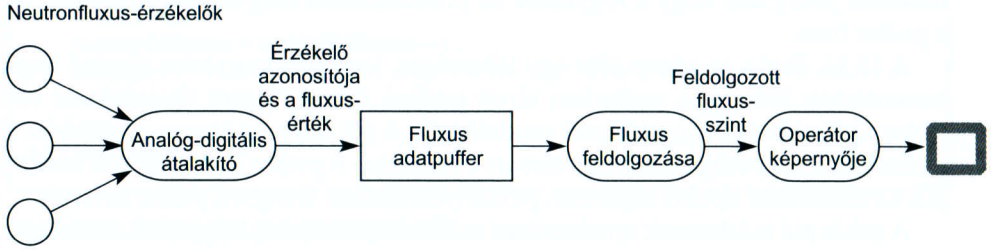
Az adatgyűjtő rendszerekben az érzékelők nagyon gyorsan generálják az adatokat, a fő problémát pedig annak biztosítása jelenti, hogy az érzékelő által leolvasott adatot még az előtt összegyűjtsük, mielőtt az megváltozik. Ez a 15.11. ábrán látható általános architektúrához vezet el minket. Az adatgyűjtő rendszerek architektúrájának lényege, hogy minden érzékelőcsoporthoz három folyamat tartozik, ezek: az érzékelőkkel kapcsolatba lépő, és szükség esetén az analóg jelet digitálissá alakító érzékelő folyamat, egy puffer folyamat és egy fogyasztó folyamat, amely a további feldolgozást végzi.

Az érzékelők természetesen különböző fajtájúak lehetnek, az egy csoportba tartozó érzékelők száma pedig a környezetből érkező adatok sebességétől függ. A 15.11. ábrán két érzékelőcsoport látható, s1–s3 és s4–s6. Az ábra jobb oldalán feltüntettem egy, az érzékelők adatainak a megjelenítését végző újabb folyamatot. A legtöbb adatgyűjtő rendszer tartalmaz megjelenítő és jelentéskészítő folyamatokat, amelyek az összegyűjtött adatokon végeznek további feldolgozást.

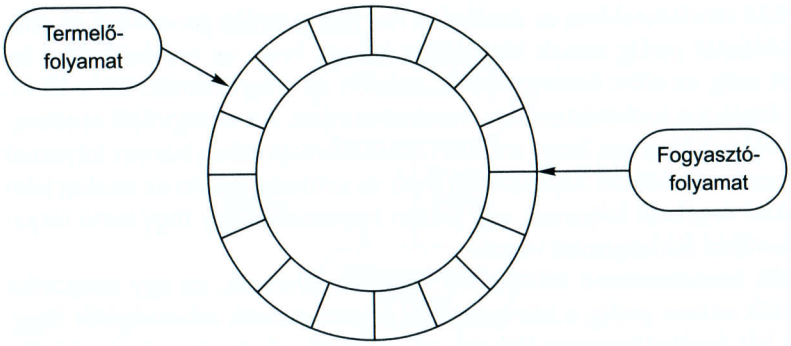
Az adatgyűjtő rendszerek egy példájaként tekintsük a 15.12. ábrán látható modellt. Ez az ábra olyan rendszert ábrázol, amely egy nukleáris reaktorban a



15.11. ábra. Az adatgyűjtő rendszerek általános architektúrája



15.12. ábra. Egy fluxusfigyelő rendszer architektúrája



15.13. ábra. Az adatgyűjtéshez használt gyűrűpuffer

neutron fluxusát figyelő érzékelőktől gyűjt adatokat. Az érzékelők adatait egy pufferben tárolják, majd onnan kerülnek feldolgozásra, az átlagos fluxusszint pedig megjelenik az operátor képernyőjén.

Minden érzékelőhöz tartozik egy folyamat, amely a bejövő fluxusszintet digitális jellé alakítja. A folyamat ezt a fluxusszintet – az érzékelő azonosítójával együtt – az érzékelő adatpufferébe tölti. Az adatfeldolgozásért felelős folyamat ebből a pufferből veszi ki az adatokat, feldolgozza azokat, majd egy megjelenítési folyamatnak adja tovább, amely kimenetet produkál az operátori konzolra.

Az adatgyűjtést és -feldolgozást magában foglaló valós idejű rendszerekben nem biztos, hogy az adatgyűjtő és a feldolgozó folyamat végrehajtási sebességei és periódusai lépést tartanak egymással. Ha fontos feldolgozási műveletet kell végezni, az adatgyűjtés gyorsabban folyhat, mint az adatok feldolgozása. Ha csak egyszerű számításokat kell végezni, akkor a feldolgozás gyorsabban történhet, mint az adatgyűjtés.

Ezen sebességbeli különbségek áthidalása céljából a legtöbb adatgyűjtő rendszer a bejövő adatok pufferelését körkörös pufferrel végzi. Az adatokat előállító folyamat (a termelő) beteszi az információt a pufferbe, míg az adatokat felhasználó folyamat (a fogyasztó) kiveszi azt onnan (15.13. ábra).

Nyilvánvaló, hogy a termelő- és fogyasztófolyamatokat a puffer ugyanazon eleméhez ugyanabban az időpillanatban való hozzáféréstől megóvandó, kölcsönös kizárást kell megvalósítani. Ezenfelül a rendszernek biztosítania kell egyrészt azt, hogy a termelőfolyamat ne próbáljon teli pufferbe információt helyezni, másfelől pedig azt, hogy a fogyasztó ne próbálkozzon meg az adatkivétellel, ha a puffer üres.

A 15.14. ábrán az adatpuffer egy lehetséges, Java-objektumként történő implementációja látható. A pufferben tárolt értékek SensorRecord típusúak, az objektum két művelettel (get és put) rendelkezik. A get művelet kivesz egy elemet a pufferből, a put pedig betesz egy elemet a pufferbe. A puffer konstruktora beállítja a CircularBuffer típusú objektum példányosításakor létrejövő puffer méretét.

A get és put módszerek synchronized módosítója azt jelzi, hogy ezek a módszerek nem működhetnek egyidejűleg. Ha az egyik meghívásra kerül, a futtatórendszer zárolja az objektumpéldányt abból a célból, hogy egyidejűleg ne lehessen

meghívni a puffer ugyanazon eleméhez hozzáférő másik módszert. Ezekben a módszerekben a wait és a notify metódushívások biztosítják azt, hogy a teli pufferbe ne lehessen elemet betenni, illetve az üres pufferből ne lehessen elemet kivenni. A wait módszer felfüggeszti az őt hívó szálat addig, amíg egy másik szál a notify módszer meghívásával meg nem szünteti a várakozását. A wait hívásakor a védelem alatt álló objektum zárja feloldásra kerül. A notify módszer felébreszt egy várakozó szálat, és újra elindítja annak végrehajtását.

```
class CircularBuffer
{
  int bufsize ;
  SensorRecord [] store ;
  int numberOfEntries = 0 ;
  int front = 0, back = 0 ;

  CircularBuffer (int n) {
     bufsize = n ;
     store = new SensorRecord [bufsize] ;
  } // CircularBuffer

  synchronized void put (SensorRecord rec ) throws InterruptedException
  {
     if ( numberOfEntries == bufsize)
        wait () ;
     store [back] = new SensorRecord (rec.sensorId, rec.sensorVal) ;
     back = back + 1 ;
     if (back == bufsize)
        back = 0 ;
     numberOfEntries = numberOfEntries + 1 ;
     notify () ;
  } // put

  synchronized SensorRecord get () throws InterruptedException
  {
     SensorRecord result = new SensorRecord (-1, -1) ;
     if (numberOfEntries == 0)
           wait () ;
     result = store [front] ;
     front = front + 1 ;
     if (front == bufsize)
        front = 0 ;
     numberOfEntries = numberOfEntries—1 ;
     notify () ;
     return result ;
  } // get
} // CircularBuffer
```

15.14. ábra. Gyűrűpuffer Java-implementációja

Kulcsfogalmak

- A valós idejű rendszerek olyan szoftverrendszerek, amelyeknek az eseményekre valós időben kell válaszolniuk. Helyességük nemcsak az általuk adott eredményektől, hanem attól az időtől is függ, amennyi idő alatt ezek az eredmények létrejönnek.
- A valós idejű rendszerek architektúrájának általános modellje az érzékelők és működtetők minden csoportjához egy folyamatot rendel. Ezenfelül szükség lehet még egyéb, koordinációt végző folyamatokra is.
- A valós idejű rendszerek architekturális tervezése általában azt jelenti, hogy a rendszert egymással együttműködő, konkurens folyamatokba kell szervezni.
- A valós idejű operációs rendszer felelős a folyamat- és erőforrás-kezelésért. Ez mindig tartalmaz egy ütemezőt, amely azért a döntésért felel, hogy melyik folyamatot kell elindítani. Az ütemezési döntésekre a folyamatok prioritása alapján kerül sor.
- A figyelő- és vezérlőrendszerek időszakosan lekérdezik a rendszer környezetéről információt szerző érzékelőket. Az érzékelőkről leolvasott adatok alapján, a működtetőknek adnak utasításokat.
- Az adatgyűjtő rendszereket általában a termelő-fogyasztó modellnek megfelelően szervezik. A termelőfolyamat egy körkörös pufferbe helyezi az adatokat, ahonnan azokat egy fogyasztófolyamat felhasználja. A puffert is folyamatként implementálva kiküszöbölhetők a termelő és a fogyasztó közötti ellentétek.

További irodalom

Software Engineering for Real-Time Systems. Mérnöki, nem pedig számítógép-tudományi szemszögből íródott, és jó gyakorlati iránymutatást ad a valós idejű rendszerek tervezéséhez. A hardveres kérdéseket jobban lefedi, mint Burns és Wellings könyve, így annak nagyszerű kiegészítése. (Cooling, J., 2003, Addison-Wesley.)

Real-time Systems and Programming Languages, 3rd edition. Átfogó, a valós idejű rendszerek minden szempontját lefedő, nagyszerű könyv. (Burns, A. és Wellings, A., 2001, Addison-Wesley.)

Doing Hard Time: Developing Real-time Systems with UML, Objects Frameworks and Patterns. Ez a könyv bemutatja, hogy az objektumorientált technikák hogyan használhatók a valós idejű rendszerek tervezésében. A hardverek sebességnövekedésével ez a valós idejű rendszerek tervezésének mindinkább járható útját jelenti. (Douglass, B. P., 1999, Addison-Wesley.)

Feladatok

15.1. Példák segítségével magyarázza el, hogy a valós idejű rendszereket általában miért konkurens folyamatok segítségével valósítják meg.

15.2. Magyarázza el, hogy az objektumorientált szoftverfejlesztés miért nem minden esetben alkalmas valós idejű rendszereknél.

15.3. Készítse el az alábbi rendszerek vezérlőszoftvereinek állapotátmenet-modelljeit.

- Automata mosógép, amely különböző programokkal rendelkezik a különféle ruhák számára.
- CD-lejátszó szoftvere.
- Telefonos üzenetrögzítő, amely rögzíti a bejövő üzeneteket, és egy LED-kijelzőn mutatja a rögzített üzenetek számát. A rendszernek biztosítania kell azt, hogy ha a telefon tulajdonosa hazatelefonál, akkor egy számsor beütésével le tudja hallgatni az üzeneteit.
- Italautomata, amely kávét készít tejjel, cukorral, illetve azok nélkül. A felhasználó bedobja a pénzt, és az automatán található gomb megnyomásával kiválasztja, hogy mit szeretne inni. Ez egy kávéport tartalmazó csészét ad ki, amelyet a felhasználó egy csap alá téve, egy újabb gomb megnyomásával felengedhet forró vízzel.

15.4. Az ebben a fejezetben leírt valós idejű rendszer tervezési technika segítségével tervezze újra inger/válasz rendszerként a 14. fejezet meteorológiai állomásának adatgyűjtő rendszerét.

15.5. Tervezzen folyamatarchitektúrát egy környezetvédelmi figyelőrendszer számára, amely egy város körül elhelyezett légminőség-érzékelőktől gyűjti be az adatokat. Az 5000 érzékelőt 100 régióra osztották. Minden érzékelőt másodpercenként négyszer kell lekérdezni. Ha egy bizonyos régió érzékelőinek több mint 30 százaléka azt jelzi, hogy a levegőminőség az elfogadható szint alá esett, be kell kapcsolni a helyi figyelmeztető fényeket. Az érzékelők a leolvasott értékeket egy központi számítógéphez továbbítják, amely minden 15 percben jelentést készít a város légszennyezettségéről.

15.6. Fejtse ki a Javának mint valós idejű rendszerek megvalósítására szolgáló programozási nyelvnek az előnyeit, illetve hátrányait. Gyorsabb processzszorok használatával mekkora mértékben tűnnének el a Javában történő valós idejű programozás problémái?

15.7. A vasúti védelmi rendszer automatikusan lefékezi a vonatot, ha az valamelyik vasúti szelvényben túllépi a megengedett sebességhatárt, vagy ha a vonat olyan szelvénybe hajt be, ahol tilos jelzés van érvényben (azaz a szelvénybe nem szabad behajtani). További részletek a 15.15. ábrán találhatók. Határozza meg, hogy a fedélzeti vasúti irányítórendszernek mely ingereket kell feldolgoznia, és azt, hogy ezekre az ingerekre milyen válaszokat kell adni.

- A rendszer a szelvények sebességkorlátjára vonatkozó információt a vasúti pálya mentén elhelyezkedő adóállomásoktól kapja, amelyek folyamatosan sugározzák a szelvény azonosítóját és a szelvényre érvényes sebességhatárt. Ugyanezek az adóállomások sugározzák a vasúti szelvényre érvényes jelzés állapotát is. A vasúti szelvények és jelzések információit 50 ms-onként kell sugározni.
- A vonat a pálya menti adóállomás 10 méteres körzetében tud információt fogadni.
- A vonat maximális sebessége 180 km/h.
- A vonaton található érzékelők a vonat aktuális sebességéről (250 ms-onként frissítve) és fékeinek állapotáról (100 ms-onként frissítve) adnak információt.
- Ha a vonat sebessége az aktuális szelvény megengedett sebességénél több mint 5 km/h-val nagyobb, a mozdonyvezető egy hangjelzést kap. Ha a vonat sebessége az aktuális szelvény megengedett sebességénél több mint 10 km/h-val nagyobb, a vonat sebességét automatikus fékezéssel a szelvény sebességhatára alá kell csökkenteni. A vonat lefékezését a kiugróan magas sebesség észlelését követő 100 ms-on belül el kell kezdeni.
- Ha a vonat egy tilos jelzéssel ellátott szelvénybe érkezik, a vasúti védelmi rendszer a vonatot állóra fékezi. A vonat lefékezését a piros lámpa észlelését követő 100 ms-on belül el kell kezdeni.
- A rendszer folyamatosan frissíti az aktuális állapotok kijelzését a mozdonyvezető fülkében.

15.15. ábra. Vasúti védelmi rendszer leírása

15.8. Javasoljon folyamatarchitektúrát a fenti rendszer számára. Dokumentálja ezt a folyamatarchitektúrát a 15.8. ábrán látható jelölésmód segítségével, egyértelműen jelölje, hogy az egyes ingerek periodikusak vagy aperiodikusak.

15.9. Ha a fedélzeti vasúti védelmi rendszerben periodikus folyamat gyűjti ösz-sze az adatokat a vasúti pálya mentén található adóállomásoktól, milyen gyakran kell ezt a folyamatot végrehajtani ahhoz, hogy a rendszer garantáltan begyűjtse az adóállomások adatait? Válaszát indokolja is meg.

15.10. Felkérik önt, hogy vegyen részt egy katonai alkalmazás valós idejű fejlesztésű projektjében, de ön ezen a szakterületen nem rendelkezik tapasztalatokkal. Fejtse ki, hogy önnek mint profi szoftvertervezőnek, mit kell tennie a munka megkezdése előtt.