Valós időben, valós világban

Dr Illés Zoltán[1], H. Bakonyi Viktória[2], ifj. Illés Zoltán[3]

{[1]illes,[2]hbv,[3]ilzo}@inf.elte.hu
ELTE IK

Absztrakt. Találkoztunk már? Olyan ismerősnek tűnsz! Tényleg, Te vagy az! Olyan gyorsan eltűntél, azt hittem valami nagyon megváltozott! Azt mondod nem tűntél el? Akkor mi történt? Fontos lett, hogy minden valós legyen! Elég a virtuális valóságból, nem ígéret kell, csak a valóság! Most, nem valamikor, valós időben, ebben a valós világban!

Kulcsszavak: valósidejűség, real-time, operációs rendszer, DOS, Linux, mikrokontroller, beágyazott rendszer

1. Bevezetés

A számítógépek egyre gyorsabbak, egyre nagyobb számítási kapacitással rendelkeznek. Ennek eredményeképpen jelentek, jelenhettek meg olyan eszközök (például a Microsoft HoloLence eszköze http://www.microsoft.com/microsoft-hololens/en-us), amelyek képesek egy virtuális valóság létrehozására. A játékok világán túl a különböző szimulációkban például repülő szimulációban, építészeti tervek láthatóvá tételében, oktatási környezetek létrehozásában adnak, adhatnak újat a felhasználók számára. Gyakran tapasztalhatjuk, hogy egy-egy ilyen alkalmazás (pl. World of Tanks) lassúnak bizonyul, akár a lövedéket röppályáján szemmel is követhetjük, de ez nem okoz katasztrófát, maximum azt jelenti, hogy a felhasználó törekszik hardver cserével, bővítéssel javítani a helyzeten!

Ugyanekkor a körülöttünk lévő világban nem mindig elegendő a korábbi, cserés, bővítéses reakció! Maga az ember is folyamatosan érzékeli a valóságot az érzékszervein keresztül, amire azonnal reagál is. Ha egy úttesten való átkelésnél rosszul becsüli meg a saját átkeléshez szükséges idejét, ami túl soknak bizonyulna egy közeledő autó érkezési idejéhez képest, akkor bizonytalan, hogy lesz-e ideje megfelelően reagálni, javítani az eljáráson!

Hasonló a helyzet a környezet eseményeinek számítógépes érzékelésével illetve azok azonnali, valós időben történő feldolgozásával! Nem elég gyorsan reagálni, fontos, hogy ez határidőre történjen! Sokan ezt kevésbé fontos kérdésnek tarthatják, hiszen látványosságban elmarad a VR (virtual reality) megvalósításoktól, ráadásul azt gondolhatják, hogy ez az csak az informatikai feladatok egy-egy rendkívül szűk felhasználási területével kapcsolatos (pl. atomerőmű szabályozás, hadászati irányítás, életmentés).

Ki kell azonban jelentenünk, hogy ma már korántsem ez a helyzet! Egyre több információt érünk el valós időben és erre az igény egyre csak növekszik! Gondoljunk csak egyszerűen a valós idejű közlekedési információs táblákra, a folyamatosan frissülő on-line hírekre, az üzleti döntéseket befolyásoló valós adatok biztosítására, a folyamatosan elérhető közösségi kommunikáció lehetőségeire (Twitter, Facebook), az akadozásmentes, minőségi online zenehallgatásra, videózásra vagy akár arra, hogy ma már senki sem vár percekig egy honlap letöltésére. Ezek mögött a szolgáltatások mindegyike mögött megtalálható a valós idejű lehetőségek eszköztára mind hardveres, mind pedig szoftveres oldalon. Az IoT (Internet of Things) elterjedésével, a smart otthonok, smart city program (http://intelligensvarosok.kormany.hu/) kiteljesedésével még inkább előtérbe kerül ez a terület, ahol az adatgyűjtés és azok azonnali feldolgozása szükségszerű.

Ezek alapján aligha vitatható, hogy a valós idejű alkalmazások készítésével kapcsolatos ismeretekkel, elvekkel minden informatikusnak tisztában kell lennie. Az informatika tanárok sem lehetnek ez alól kivételek, hiszen különböző mérések – például a szabadesés – számítógépes megvalósításával kézzelfoghatóbbá teszik a diákok számára a természettudományos összefüggéseket. Kiváló példákat láthattunk dr. Piláth Károly: Egy kis informatika a fizika órán, 2014, InfoEra előadásában erről a témáról.

2. Valós idejű rendszerek

A valós rendszerekben, környezetekben, az életből vett példák mintájára, egy feladatra, kérdésre adott válasznak „megfelelő" időn belül meg kell születnie! Konkrétabban egy valós idejű számítógépes rendszerben egy bekövetkező kérésre, eseményre adott időn belül eredményt, választ kell adni! A kérdés csak az, hogy mennyi is az adott idő? Erre, ahogy az életben is sok kérdésre, nem mindig van pontos válasz! Ha a rendszerrel szemben nem elvárás, hogy pontosan, határidőn belül fejezze be a feladatát, hanem csak törekednie kell rá, hogy lehetőség szerint időre végezzen, akkor lágy valósidejű (soft real-time) rendszerről beszélünk. Ellenkező esetben, ha a határidő be nem tartása katasztrófális következményekkel jár(hat), akkor kemény valósidejű (hard real-time) rendszerről beszélünk.

Mik azok a főbb jellemzők, amelyek leginkább elemei, eszközei és egyben befolyásolják a valós idejű rendszerek működését?

- Elsőként nyilvánvalóan magát a feladatot megvalósító kódot említhetjük, hiszen ha ennek a végrehajtási ideje hosszabb, mint amekkora időintervallumon belül a válasznak meg kell születni, akkor eleve nem járhatunk sikerrel.
- Elengedhetetlenül fontos a rendszeróra, a különböző időzítők szerepe! Nem elégséges ma már a hozzávetőlegesen ezredmásodperc pontosságú időzítők használata, helyettük nanosec pontossággal tudunk időt mérni!
- Hasonlóan lényeges a megszakítások, jelzések továbbítása, kezelése, hiszen ezek kezelése komolyan befolyásolhatják a végeredményt.

2.1. Találkoztunk már?

Olyan ismerősnek tűnsz! „Hanyas vagy?" 81-es! Tényleg, Te az vagy!

Mind ismerjük (?) a DOS operációs rendszert (több változata volt,IBM PC DOS, MS-DOS, Free-DOS), ami egyfelhasználós, egyfeladatos rendszerként definiálható. Nyilvánvaló, hogy ebben az esetben egy valós idejű alkalmazás készítésénél nem kell figyelembe venni más egyidejűleg futó alkalmazások zavaró hatásait. A határidők tartása magán az alkalmazás gyorsaságán és az időzítők, megszakítók működési gyorsaságán múlik. [1,2]

Gondoljunk bele, hogy mire is kell figyelni, mondjuk egy a témakörre jellemző, valósidejű periodikusan mérő alkalmazás kivitelezésénél. Kérdés, hogy a CMOS valós idejű óra pontossága megfelelő-e az adott feladathoz – az óra megszakítást generál 1/1024 másodpercenként. Nyilván ez az érték illetve ennek többszörösei jól használhatóak. Mint tudjuk, egy megszakítás bekövetkeztekor az aktuális folyamat futása megszakad, a rendszer a további megszakításokat maszkolja, lefut az adott megszakításhoz definiált eljárás, majd a félbehagyott program folytatódik. Akkor tudja tartani a határidőt a rendszer, ha ez a folyamat az adott időn belül végrehajtható.

A DOS könyvtári szolgáltatásai között rendelkezésünkre állt egy speciális 1Ch hívás, ami alapértelmezésként 55 ms után megszakítást generál! Ennek a kezelése összesen egy IRET utasítást tartalmazott, azaz arra volt felkészítve, hogy saját rutint illesszünk ide be!

A program maga készüljön bár assembly vagy C nyelven, mindenképpen gépi kódra lefordított állapotban fut. Mivel ismerjük az egyes gépi kódú utasítások végrehajtásának idejét, így az utasítások összvégrehajtási ideje kiszámítható, megbecsülhető. A DOS tehát akár kemény valósidejű rendszerek (hard real time) készítésére is alkalmas volt.

2.2. Valós idejű, időosztásos operációs rendszerek

Olyan gyorsan eltűntél! Azt mondod nem tűntél el? [1] Csak újra fontos lett, hogy minden valós legyen!

Az informatika fejlődésével az operációs rendszerek között megjelentek majd elterjedtek a több felhasználós, többfeladatos, preemptív időosztásos rendszerek. [4] Ezek között voltak olyanok is, amelyeket a beágyazott rendszerek számára fejlesztettek ki (QNX), ahol a valós idejű jellemzők, a gyorsaság, a kis méret volt megjelölve célként! Ugyanakkor voltak az általános célú operációs rendszerek (LINUX, Windows) amelyek mindenre figyeltek, csak a valós idejű feladatok támogatására nem! Persze ez nem jelentette azt, hogy nem volt semmilyen valós idejű alkalmazás készítési lehetőség, csak azt, hogy ehhez az operációs rendszer nem igazán adott segítséget!

Az utóbbi időben a valós idejű feladatok és igények egyre inkább előtérbe kerültek, ezért ennek támogatására az általános célú operációs rendszereket is igyekeztek felkészíteni valósidejű feladatok ellátására.

A legelterjedtebb általános célú valós idejű operációs rendszer, amely valós idejű alkalmazások futtatására is alkalmas, a Linux (a 2.6.3 verziótól) a Real Time Modullal kiegészített változata. (Jelenleg a Kernel 3.12-es verzióját használjuk.) [3]

Mivel többfelhasználós, többfeladatos operációs rendszerről van szó, meg kellett oldani azt a problémát, hogy a valós idejű feladatok egyidejűleg futhassanak a normál folyamatok mellett úgy, hogy az előbbiek megtarthassák az időérzékenységüket. Multitask rendszerek kulcsszereplője a feladat (folyamat) ütemező! Prioritásos rendszerben, annak figyelembevételével a folyamatok az ütemező „felügyelete mellett" futnak! A klasszikus prioritási rendszer bővült a valós idejű prioritásokkal! [8,6,7]

Mivel ma a szerverek többsége több processzorral rendelkezik, a feladatot nehezíti a feladatok egyenletes elosztása a CPU-k között. (A LINUX rendszerben egy-egy feladatot manuálisan is hozzárendelhetünk bizonyos CPU-khoz CPU-set-ekhez.)

Az ütemező feladata, hogy az erőforrásokat biztosítsa a folyamatok számára. A folyamatok között lehetnek normál illetve valós folyamatok is. A cél az, hogy a normál folyamatoknak minél „igazságosabban" biztosítsa a CPU hozzáférést (ma CFS – Completely Fair Scheduler), a valós folyamatoknak pedig prioritásukat figyelembe véve kiemelt futási lehetőséget biztosítson (Round Robin vagy FiFO ütemezési módszer szerint).

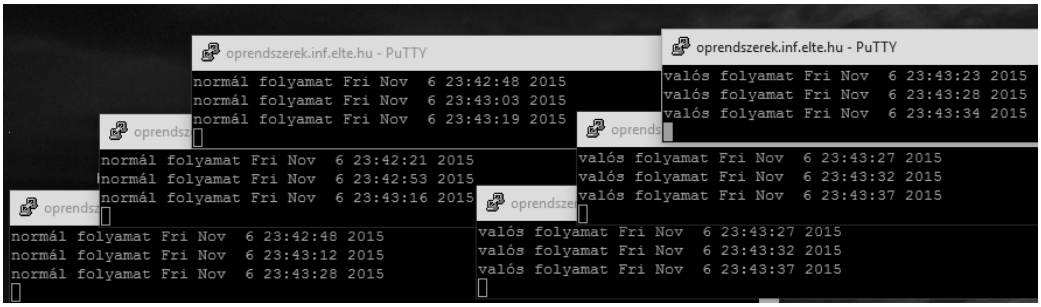
Az ütemező valós idejű affinitásának megjelenése mellett még egy jellemzőt kell megemlíteni! Ez pedig a megszakítás, jelzés küldés módosulása! A klasszikus jelzések mellett valós idejű jelzések kerültek bevezetésre! Ezek POSIX ajánlás szerint többféleképpen implementálhatók, SIGRTMIN…SIGRTMAX intervallummal biztosítják az elsőbbségi, ajánlott küldemények célbaérkezését!

Az ütemező munkáját szemléltetjük a következő kis mintakóddal és futtatásával:

```
#include <stdio.h>
#include <time.h>
int main(int argc,char** argv){
 int i=0,d=0;
```

```
 while (1)  {
  i=i+1; i=i%10000; //számolunk
  if (i==0){
   if (++d==50000) { //10000*50000 végrehajtás utáni idő
    time_t t=time(NULL);
    printf("%s %s",argv[1],ctime(&t));
    d=0;
   }
  }
 }
return 0;
}
```

A programot 3 normál és 3 valós idejű folyamatként is elindítjuk (külön-külön terminál ablakban) egy 4 processzoros gépen. Valós folyamatot csak rendszergazda jogosultsággal, a chrt parancs segítségével indíthatunk parancssorból. A valós idejű folyamatoknak egy-egy processzszort dedikál az ütemező, a normál folyamatok pedig osztoznak a maradék CPU-n, így ezek jóval lassabban hajtódnak végre, ahogy az 1. ábrán is látszódik!



1. ábra 4 processzoros gépen, LINUX RT 3-3 normál és valós folyamat futási ideje

Ahogy azt korábban említettük egy valós idejű alkalmazásnál fontos az időzítők, jelzések megvalósítása is. A LINUX lehetővé teszi, hogy időzítőket alkalmazzunk, nano másodperces pontossággal, amely vagy a valós időre vagy pedig az adott folyamat CPU-ban töltött idejére vonatkozik. Ezt mutatja a következő kód:

```
…
timer_t timerid;
timer.it_interval.tv_sec=0; /* ismétlődés */
timer.it_interval.tv_nsec=1;
 …
sev.sigev_notify=SIGEV_SIGNAL;
sev.sigev_value.sival_ptr=&timerid;
sev.sigev_signo=SIGRTMIN; //a jezés fajtája
…
timer_create(CLOCK_REALTIME,&sev,&timerid);
timer_settime(timerid,0,&timer,NULL );
```

Az időzítő lejártakor egy jelzést küld az ezt kérő folyamatnak. A jelzés lehet normál jelzés vagy valós idejű jelzés. Az operációs rendszer ezeket különbözően kezeli. A normál jelzések esetében az ugyanolyan típusú (a mintában SIGUSR1) jelzések közül csak egyet tart meg, míg a valós jelzések mindegyike végrehajtódik. A következő kód részlete ezt a tulajdonságot mutatja be.

```
…
volatile int  normal=0;
volatile int valos=0;
void handler_normal(int signumber){     normal++; }
void handler_valos(int signumber){     valos++; }
int main(){
 signal(SIGUSR1,handler_normal);
 signal(SIGRTMIN,handler_valos);
 int i;
 pid_t child=fork();
 if (child>0)  {
    wait(NULL);
    sleep(3);
    printf("%i normál %i valós jelzés érkezett\n",normal,valos);
 }
 else {
  for (i=0;i<5;i++){
   kill(getppid(),SIGUSR1);
   kill(getppid(),SIGRTMIN);
  }
 }
 return 0;
}
```

Az eredmény a 2. ábrán látható:

```
>gcc -lrt sig.c -o signal
>./signal
1 normál 5 valós jelzes érkezett
```

2. ábra Normál és valós szignálok kézbesítése

2.3. Beágyazott rendszerek, mikrokontrollerek

„Fontos lett, hogy minden valós legyen! Elég a virtuális valóságból, nem ígéret kell, csak a valóság! Most, nem valamikor, valós időben, ebben a valós világban!”

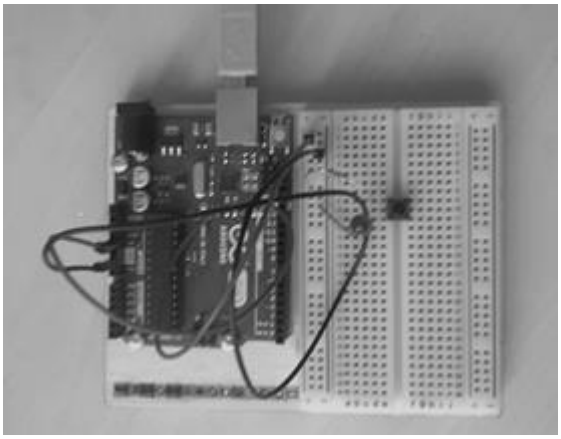
Napjainkban a számítógépes processzorok 90%-a már beágyazott rendszerekben kerül felhasználásra! (http://bit.ly/20AJQfD) Elterjedőben vannak a mikrokontrollerek, amelyek olcsóságuk miatt mindenki számára hozzáférhetőek, a CPU mellett memória és alap külvilági kapcsolatokat biztosító periféria elemekkel (IO lábak, soros port, ADC, stb), időzítőkkel rendelkeznek. Azt mondhatjuk, hogy egy ilyen integrált áramkör egyben egy komplett számítógép!

Egyre többen vásárolnak maguknak Arduino vagy Raspberry készleteteket, amelyekkel egy sor mintaalkalmazás, esettanulmány megvalósítását elvégezhetjük (fénymérés, diódák, lámpák, motorok kapcsolgatása), illetve kiegészítők megvásárlása után általános célú intelligens eszközként tetszőleges fejlesztésre alkalmazható!

Ma ezen eszközöknek alkalmazásaival, leggyakrabban megvalósított mintáival talán az intelligens otthon különböző eseteiként találkozhatunk.

Természetesen ezek az eszközök kisebb erőforrásokkal rendelkeznek, de a megcélzott feladatok se olyan robosztusak, mint egy általános célú számítógépben. Nincs szükség például az előbbiekben emlegetett professzionális SuSe Linux Enterprise OS Real-Time modulos kiterjesztésére. Azt tapasztalhatjuk, hogy visszatérünk a gyökerekhez, ezeken a területeken újra a jól ismert OS-ek szűkebb funkcionalitásai jelennek meg!

Az alábbi képen (3. ábra) egy Arduino Uno board látható, Atmer 328P vezérlővel, amit ha csomagban vásárolunk meg, egy sor valós alkalmazás készítésére lesz alkalmas! Használhatunk C fejlesztői környezetet, amit akár Visual Studio 2015 alá is installálhatunk. Ami a célszámítógép jellegből adódik, tapasztalhatjuk, hogy sokkal fontosabbak a periféria kezelő könyvtári szolgáltatások használata, ismerete, mint a formális C nyelvi keret! [5]



3. ábra Arduino Uno Board

Egy lényeges változást jelent ez az új világ! Míg korábban volt egy számítógépünk, majd később a gyerekek is kaptak egyet, hogy ne rontsák el a szülők gépét, addig ma lassan nem tudunk olyan eszközt mondani a környezetünkből ami ne számítógép lenne! Természetesen mindegyik internetre csatlakoztatva! IoT!

Ide kívánkozik, hogy a Windows-os világ is nyújt támogatás a valós idejűség terén is. Korábban a Windows CE-t használhatták a felhasználók erre a célra, de 2015. augusztus 11-én megjelent a Windows 10 IoT verzió is (http://bit.ly/1iNEfAu), ami azt tesz lehetővé, hogy elvileg ugyanazok az alkalmazások fussanak mind a számítógépen, mind a mobil eszközökön, mikrokontrollereken. Igaz a legszűkebb verzió nem tartalmazza a shell-t, de az eszközre feltöltött az alkalmazás megfelelően működik.

3. Összegzés

Örvendetesen bővült az utóbbi években a számítógépes alkalmazások, világok köre, gondoljunk csak az elsősorban „Robotika” kulcsszó köré csatlakoztatható eszközökre! Ezek általában rendelkeznek valamilyen programozási lehetőséggel, sokszor elég egy grafikus felületen egy programlogikát összeállítani és már kész is a „játék”!

Mára azonban ez a játék már nem játék, hanem maga a valóság! Egyrészt velünk van mikrokontroller formájában a valós célfeladat eszköztárunk, másrészt kedvenc operációs rendszerünk is biztosítja számunkra a valóságot!

Jó újra Veled! Dolgozzunk valós időben, valós világban!

Irodalom

1. Illés Zoltán: *Valós idejű mérések megvalósítása nagyenergiájú ionbesugárzásokhoz*, PhD értekezés, 2001

2. Illés Zoltán, Havancsák Károly: *Real-Time Computer Control under DOS-Windows Operating system*, Konferencia helye, ideje: Budapest, Magyarország, 1998.07-1998. 8 p.
3. *SUSE® Linux Enterprise Real Time User's Guide*, 4. fejezet http://bit.ly/1Rlt0LP (utoljára megtekintve: 2015.11.5.)
4. Andrew S. Tanembaum, Albert S. Woodhull: *Modern Operating Systems*, 2014. ISBN-13: 978-0133591620
5. Dennis M. Ritchie, Dennis M. Ritchie*: The C Programming Language 2nd Edition*, AT&T laboratories, New Jersey ISBN-13: 978-0131103627
6. Illés Zoltán, Heizlerné Bakonyi Viktória, Horváth László, Nagy Tibor, Lutár Patrícia: *Számítógépes alapismeretek II*., 5. lecke ELTE Informatika Kar, http://www.tankonyvtar.hu (2012) (utoljára megtekintve: 2015.11.5.)
7. Illés Zoltán: *Operációs rendszerek 5. előadás*,http://bit.ly/1MAdQ7m (utoljára megtekintve: 2015.11.5.)
8. Robert Love: *Linux kernel developemen*t, Third Edition, Addison-Wesley, (2010)