Gyártófolyamatok automatizálása Linuxszal

Avagy hogyan kísérhetünk figyelemmel valós időben több gyártósort? A problémánkat saját magunknak oldottuk meg a Linux segítségével.

Egy termeléssel foglalkozó vállalat csak akkor termel hasznot, amikor működnek a gyártósorok, ezért létfontosságú, hogy a gyártószintről származó információk kellő időben rendelkezésre álljanak. A vállalatunk méretével együtt növekedett a bonyolultsága is, így rövid idő alatt kinőttük a termelés ellenőrzésére hivatott manuális, papír alapú módszereinket.

Cégünk, a ***Midwest Tool & Die*** elektronikus csatlakozóvégeket sajtol, és műanyag alkatrészeket formáz az autógyártás, elektronikai és fogyasztói ipar számára. A gyártófolyamatainkban rengeteg adat keletkezik. A nagysebességű préseink percenként akár 1200 alkatrész legyártására is képesek, és minden egyes darabnak megfelelőnek kell lennie. Minden gyártott alkatrésznek megvizsgáljuk a kritikus méreteit, folyamatosan figyelemmel kísérjük és ábrázoljuk a minőséget, az adatokat pedig a nyomon követhetőség érdekében archiváljuk.

Miért fontos az automatizálás?

A gyártófolyamataink továbbfejlesztéséhez szükségünk volt az összes adat kezelésére. A fő célunk az volt, hogy növeljük az üzemidőt és minél jobban megértsük az állásidő okait. Ezen felül reménykedtünk abban, hogy a költségeket is nyomon követhetővé és ellenőrizhetővé tehetjük, csökkenthetjük a papírmunkát és elkerülhetővé válik az emberi adatbevitelből származó hiba.

E célok alapján körvonalazódtak az új rendszerrel kapcsolatos elvárások. Elsődleges elvárás volt az adatok összegyűjtése a sokféle gépvezérlő egységből, érzékelőkből, automatikus vezérlőegységekből, a programozható logikai vezérlőkből (***PLC***) és az operátorként dolgozó munkatársaktól. A rendszernek megbízhatónak kellett lennie és képesnek arra, hogy a legnagyobb termelési sebesség mellett is összegyűjtse az adatokat. A következő elvárás az volt, hogy a rendszer meg is feleltesse az így begyűjtött adatokat, vagyis párbeszédképesnek kellett lennie a cég ***PostgreSQL*** adatbázisaival. A termelési adatok és a folyamat állapota a ***PostgreSQL***-nek adódik át megjelenítés és jelentéskészítés céljából.

Az új automatizáló rendszernek felhasználói felülettel is rendelkeznie kellett, amelyen a gépkezelők és a karbantartó személyzet naplózhatja a saját tevékenységét. Az állásidők és ezek okai rögzítésre kerülnének és továbbítódnának a vállalati adatbázis felé. Ezzel a megoldással kiváltható lenne a papír alapú naplózásra és manuális adatrögzítésre fordított munka.

1. ábra Két SmartPress gyártási automatizáló rendszer használat közben. Minden rendszer egy PC-ből, egy DAQ-kártyából, elektronikus leválasztó rendszerből, tartalék akkumulátorból és vonalkód-nyomtatóból áll.

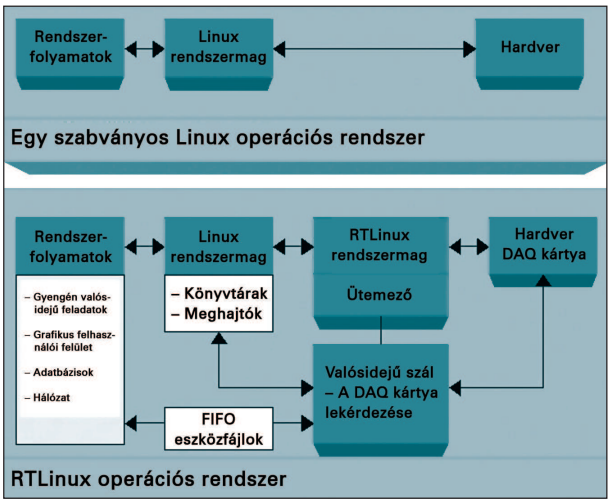
Végül a rendszernek rugalmasnak s könnyen frissíthetőnek kellett lennie továbbá alkalmasnak kellett lennie új gyártósorokra való átültetésre és a rendszerbemenetek megváltozásának kezelésére.

Miért a Linuxot választottuk?

A fent vázolt elvárások tükrében több megoldást is megvizsgáltunk. Az ipari ***PLC***-k megbízhatóan össze tudnák gyűjteni az adatokat, a hálózati megoldásaik azonban évtizedekre megrekedtek bizonyos nem szabványos, szabadalmilag védett eljárások szintjén. Az ***Ethernet*** kapcsolat ugyan elérhetővé vált, de az ilyen rendszerek drágák.

A felhasználói felület jellemzően a gyártófüggő megjelenítő hardverre készül. Minden gyártó kifejleszti a saját szabadalmazott fejlesztői felületét. Látható tehát, hogy az értékelés minden pontján megjelenik a gyártótól való függés.

A következő próbálkozásunk egy adatgyűjtő (***data acquisition, DAQ-***) kártyával felszerelt PC volt. Korábban egy táskagépet használtunk egy ***DAQ***-kártyával, ***Microsoft Windows***-t és ***Aglient VEE***-t. Ez az együttes jól működött az adatok gyors összegyűjtésének terén, ehhez csak kevés programozási munkára volt szükség. Ezzel a rendszerrel az adatok továbbítása az adatbázisunkhoz csak a ***Windows OLE***



2. ábra Az RTLinux-rendszerfolyam

felületén keresztül volt lehetséges. Írhattunk volna egyedi alkalmazásokat de a szabadalmazott felület a gyártóhoz kötött volna minket. A ***National Instruments*** szintén ajánlott teljes ***DAQ***-csomagot a PC-hez, de csak felár fejében.
Az igényeinknek legjobban megfelelő megoldás is PC-ből és DAQ-kártyából áll, az eltérés az ***RTLinux*** használata, amely egy ***Linuxra*** épülő valós idejű operációs rendszer. Ezzel korlátozhattuk a gyártóhoz való kötődést és ingyen létesíthettünk kapcsolatot a ***PostgreSQL*** adatbázissal és a ***TCP/IP*** hálózattal. A valós idejű operációs rendszer a ***PLC*** megbízhatóságát kínálta anélkül, hogy cserébe az összeköttetésekről le kellett volna mondanunk. Nem utolsó sorban pedig a felhasználó felületet is a saját tetszésünknek megfelelő nyelven készíthettük el. A nyílt forráskódú eszközök használatával rugalmas és könnyen frissíthető alkalmazásokat hozhattunk létre.

A SmartPress rendszer

A számítógépek, amelyeket az adatbegyűjtés és az adatok kezelésének feladatára választottunk, a manapság kapható gépekhez képest lassúak. Lehetőségünk adódott ugyanis a régi irodai gépeink újrahasznosítására a gyártási szinten. Ezek a 400 MHz-es ***Celeron*** processzoros gépek elég gyorsak ahhoz, hogy megfelelően elvégezzék a rájuk bízott feladatokat anélkül, hogy akadályoznák az adatok begyűjtésével kapcsolatos magas szintű elvárásainkat. Az általunk alkalmazott rendszerre induláskor a ***Red Hat 7.3*** rendszercsomag került a 2.4.18-as rendszermaggal.

Az RTLinux és a Linux rendszermagok

A rendszermag választja el a felhasználói szintű feladatokat a rendszer hardverétől. A szabványos ***Linux*** rendszermag minden egyes felhasználói kérés számára időszeleteket oszt ki és képes ezek lejártakor az adott feladat felfüggesztésére. Ez ahhoz vezethet, hogy a kritikus fontosságú feladatokban késést okozhatnak a rendszer nem kiemelt fontosságú folyamatai. Léteznek olyan parancsok, amelyekkel befolyásolhatjuk a ***Linux*** feladatütemező működését, bár a 2.4-es rendszermagban ezeknek a paramétereknek az átállítása sem eredményez igazi valós idejű rendszert. A 2.6 rendszermagban már fejlettebbek a valós-idejű képességek, de még ez sem elégíti ki egy ***szigorúan valósidejű rendszerrel*** (***hard real-time system***) szemben támasztott követelményeket.
Rengeteg nagyszerű kiadvány jelent már meg az ***RTLinux-szal*** kapcsolatban, sok ezek közül ***Michael Barbnovtól*** és ***Victor Yodaikentől*** származik, akik először valósították meg az ***RTLinuxot*** még 1996-ban, és azóta is fejlesztik. A ***Finite State Machine Labs, Inc. (FSMLabs)*** egy a tulajdonukban lévő szoftvercég, amely karbantartja a programot. Az évek során kidolgoztak olyan továbbfejlesztéseket, amelyeket az ***RTLinux*** kereskedelmi változataiba építettek be, de továbbra is biztosítanak egy ingyenes változatot ***RTLinux/Free*** néven, amely a ***GNU GPL*** és az ***Open RTLinux Patent Licence*** (***nyílt RTLinux szabadalmi licenc***) alapján használható. A projektünkhöz az ingyenes változatot használtuk, amelyhez nem jár támogatás az ***FSMLabs***-tól.
A ***Dinil Divakarannak*** köszönhető ***RTLinux HOWTO***-ban megtaláltuk az ***RTLinux*** telepítéséhez és futtatásához szükséges információ túlnyomó részét.

Az RTLinux-rendszerfolyam

Egy normál ***Linux*** rendszerben ha írunk egy függvényt, amely az adatgyűjtő kártyáról érkező adatokat meghatározott időnként beolvassa, nem kapunk kielégítő eredményt. Egy ilyen rendszer nem tudja garantálni a kapott ütemezési elsőbbséget. Amint a 2. *ábrán* látható, a szabványos ***Linux*** operációs rendszerben minden rendszerfolyamat a hardvertől elválasztva működik. Ez nem is lenne gond, ha az adatbeolvasó folyamatunk lenne az egyetlen, amit a rendszer éppen végrehajt. A mi projektünknek azonban felhasználói térben futó programra van szüksége, és garanciára, hogy az érzékelők bemeneteit minden ezredmásodpercben beolvashatja. A szigorúan valósidejű rendszerek biztosítani tudják, hogy az érzékelők bemeneti adatai egyszer sem vesznek el. Az ***RTLinux*** operációs rendszerben, amely szintén látható a 2. ábrán, a valós idejű feladat el van választva a többi rendszerfolyamattól, és modulként kerül megvalósításra. A modul közvetlen hozzáférést élvez a hardver és a ***DAQ***-kártya meghajtói felé, és a rendszer többi része felett is megkapja a szükséges prioritást. A modult egy adott feladat megoldására írják, amely megbízható eredményeket biztosít, és ezeket a ***FIFO*** meghajtó-fájlokon keresztül adja át a felhasználónak. A fejlesztők törekedtek arra, hogy a modul kódja minél egyszerűbb maradjon, csak azokat a feladatokat oldották meg vele, amelyeknek mindenképpen valós időben kell zajlaniuk. Az egyik ***FIFO*** meghajtó-fájlhoz hozzákapcsolva egy kezelőfüggvényt lehetőség nyílik arra, hogy az operátor kezelőprogramja ezen keresztül végezze az irányítást. Az ***RTLinuxnak*** ez a felépítése biztosítja, hogy a rendszermag másodlagos folyamatok futtatása miatt ne késleltethesse a fontos modulfeladatok végrehajtását.

Az adatgyűjtő szálak

Egyedül a digitális bemenet állapotát kell a szigorú valós idejű feltételekkel figyelnünk, ezért a valós idejű függvényünk kis méretű maradhatott. Az adatbemenetek lekérdezését megkönnyítette, hogy a ***United Electronics Industries*** a ***DAQ***-kártyáihoz ***RTLinux*** meghajtóprogramot is adott. Az ***ADLINK Technology, Inc. DAQ***-kártyáját szintén az ***RTLinuxhoz*** készült meghajtókkal teszteltük, a beállítással

1. lista Egy valós idejű modul kódjának váza

```
#include <pthread.h>
#include <rtl_fifo.h>
#include <rtl_core.h>
#include <rtl_time.h>
#include <rtl.h>
#include <rtl_fifo.h>

pthread_t thread_variable;

void thread_name(void)
{
    Struct Sched_param p;
    p.sched_priority = 1
    pthread_setschedparam(pthread_self(),
                          SCHED_FIFO, &p);
    pthread_make periodicnp(pthread_self(),
                            getrtime(),
                            ➥1000000);
    while(1) {
        // Real Time Task Code
        // Poll Data input lines, count low to
        // high transitions
        rtf_put() // Counts to be transferred by
                  ➥FIFO
        pthread_wait_np();
    }
}

int handler_function(){
    // Code tied to the handler FIFO
    // Variables for counting above are cleared
    // out
}

int init_module(void)
{
    ififo_status =  rtl_create(unsigned int fifo,
                               int size)
    pthread_create(&thread_variable, NULL,
                   thread_name, NULL);
    rtf_create_handler(FIFO_Number,
                       &handler_function)
}

int cleanup_module(void)
{
    rtf_destroy(unsigned int fifo)
    pthread_cancel(thrad_variable)
}
```

nem volt gondunk. Nem sok cég kínál ilyen szolgáltatást, jóllehet a ***Comedi Project*** a nyílt forrású meghajtók, segédprogramok és programkönyvtárak révén egy másik lehetőséget is kínál az adatgyűjtés megvalósítására.

A valós idejű feladatot egy betölthető modul formájában írtuk meg, amelynek legalább két függvénnyel kell rendelkeznie: Az egyik az `init_module` függvény, amelyet a modul rendszermagba történő beillesztése előtt hívunk meg, a másik a `cleanup_module`, amelyet közvetlenül az eltávolítás előtt hívunk (*1. lista*).

Miután a modul alapszerkezete elkészült, szükségünk volt a modulban egy szál létrehozására, amely a valós idejű feladatot végzi el. A szálat az `init_module` belsejében hoztuk létre, és úgy állítottuk be, hogy a megfelelő ***RTLinux*** prioritásokkal fusson. A megfelelő prioritás és futási sebesség beállítása a szál számára a valós idejű feladat létrehozásának fontos lépését jelentette.

A FIFO meghajtófájlok

A kiszámítható időzítésű futtatás megvalósításához szükség volt valós idejű memóriára az adatátvitelhez. A felhasználói szintű folyamatnak hozzá kellett férnie az összegyűjtött adatokhoz és a valós idejű folyamat vezérléséhez. A valós idejű ***FIFO***-k olyan várakozási sorok, ahonnan a ***Linux*** folyamatok adatokat olvashatnak ki és írhatnak bele vissza. A valós idejű ***FIFO*** meghajtók fordítása az ***RTLinux*** telepítésekor történik, a létrehozása pedig a valós idejű modulok inicializálásakor. Ekkor egy kezelőprogram hozható létre és köthető az egyik ***FIFO*** meghajtóhoz. A kezelőt úgy állíthatjuk be, hogy akkor fusson le, amikor a felhasználói felületről egy 1-es érték érkezik a kezelő ***FIFO***-ba mint a szükséges modul vezérlője. A modult a rendszermagba valós idejű folyamatként való telepítés céljából hoztuk létre. A valós idejű modul bonyolult része a keretrendszer létrehozása volt. A valós idejű feladat magától értetődő volt, bár igen hosszú, ezért ide csak a valós idejű modul vázát illesztettük be (1. lista). Láthatjuk, hogy milyen egyszerű a valós idejű elvárásokat megvalósító kód.

A felhasználói felület

A valós idejű folyamat üzembe helyezésének következő lépése az alkalmazás felhasználói felületének létrehozása volt. A programunkban a grafikus felület volt annak a számos feladatoknak az egyike, amelyeknek nem kellett szükségszerűen teljesíteniük a szigorú valós idejű kikötéseket. Nem kellett különösebben törődnünk a rendszererőforrásokkal, mivel a valós idejű modulunk a már megvalósított valós idejű környezetünkben fog futni. A választásunk a ***KDevelop IDE*** és a ***Trolltech Qt Designer*** grafikus felület készítő eszközére esett. A ***Qt Designer*** lehetővé tette egy olyan grafikus felület kifejlesztését, amely a ***KDevelopban*** elkészített program függvényeivel képes volt a jelekkel és csatolókkal megvalósított kapcsolattartásra. Ennek a két eszköznek az együttes használatával létrehozott felület tökéletesen megfelelt a programunk számára. Rövid időn belül képesek voltunk egy felhasználóbarát felület létrehozására.

A program két forrásból képes a rendszer számára információkat fogadni: a ***DAQ***-kártya felől jövő digitális bemenetről és a grafikus felületen keresztül a kezelőszemélyzettől. Ennek a két forrásnak az egyesítésére volt szükség ahhoz, hogy megőrizhessük az adatok sértetlenségét. Például a kezelő a munkafolyamathoz beírja a gyártandó alkatrész számát, a futás alatt összegyűjtött adatok pedig ehhez az alkatrészszámhoz rendelődnek. Ez helyettesíti a felhasználó korábbi kötelezettségét a nyilvántartások kézzel történő kitöltésére vonatkozóan.

Az adatok kezelése

Az adatok összegyűjtése csak a rendszer és a program első lépését jelentette. A legfontosabb az volt, hogy az információkat használhatóvá tegyük a cég minden osztálya számára. A tervezési, beszerzési és karbantartó osztály csak néhány példa azokra a helyekre, ahol ezeknek az információknak jó hasznát vehetik.

A *SmartPress* program az összegyűjtött adatokat egy *PostgreSQL* adatbázisban helyezte el. A *PostgreSQL* rendszer képezi a vállalati szintű háttéradatbázisunkat is, így a jövőben kifejlesztendő alkalmazás egész vállalati szinten láthatóvá teszi majd a *SmartPress* adatait.

Összegzés – „csináld magad" információtechnológia

A *SmartPress* rendszerünk a tesztelőszobából kikerült a termelési szintre. Visszatekintve elmondhatjuk, hogy sikerült létrehoznunk egy rugalmas gyártásellenőrző rendszert. A *Linux* használatával egy olyan bővíthető alkalmazást kaptunk, amely a cég igényeitől függően szabható testre. Lehetőség van a *SmartPress* új gyártófolyamatokra történő bevezetésére is. A rendszer egyszerűen frissíthető, a jövőben tervezzük is a program új rendszermagra történő átültetését. A frissítéshez valószínűleg a *Fedora* magot fogjuk *Linux*-alapként használni.

A *SmartPress* alacsony hardverköltségei fontos szempontot jelentenek számunkra, mivel minden egyes préssor számára egy-egy külön rendszert telepítünk. A költségeket személyi számítógépek és a *Linux* által támogatott *DAQ*-kártyák használatával tudtuk alacsony szinten tartani.

A programfejlesztésünk szintén olcsó volt, az idő, amelyet *Ryan Walsh* ezzel a projekttel töltött, körülbelül annyi, mint amennyit egy kereskedelmi vezérlőnyelv megtanulásával és az abban történő fejlesztéssel kellett volna eltöltenie. *Ryan* mostanra teljesen otthonosan mozog a rendszermag-modulok, a valós idejű operációs rendszerek, a *PostgreSQL* és a grafikus felületek fejlesztése terén, ezek a készségek pedig sokkalta használhatóbbak, mint egyetlen gyártó programozási nyelvének elsajátítása. Számunkra a „csináld magad" módszer választása azt eredményezte, hogy alacsonyabb költséggel, gyártóhoz való kötöttség nélkül és fejlesztői ismereteink gyarapításával sikerült elérnünk a céljainkat.

Linux Journal 2004. december, 128. szám

Craig Swanson (craig.swanson@slssolutions.net) hálózattervező és Linux tanácsadó az SLS Solutions cégnél. A Midwest Tool & Die cégnél Linux programok fejlesztésével foglalkozik. 1993 óta használ Linuxot.

Ryan Walsh (ryan.walsh@midwest-tool.com) a Midwest Tool & Die hálózati mérnöke. Szabadidejében repülőgépekből kiugrálva élvezi a szabadesés élményét.