Universitatea Politehnica Bucuresti
Facultatea de Electronica, Telecomunicatii si Tehnologia Informatiei

Implementarea mecanismelor de I/E

Gestionarul Plug-and-Play in Windows

Angelica Negrila

431 A

- 2012 -

Ce este plug and play?

Plug-and-play (PnP) este o tehnologie moderna care permite descoperirea si configurarea automata a dispozitivelor fizice atasate unui computer, oferind facilitati pentru arbitrarea conflictelor legate de accesul la resursele partajate ale sistemului. De asemenea, permite utilizarea dispozitivelor atasate PC-ului de catre utilizator fara a necesita interventia acestuia in procesul de configurare. In esenta, identificarea resurselor se bazeaza pe citirea unui tuplu de tip (vendor ID, deviceID) si incarcarea componentelor software necesare pentru a realiza managementul resursei atasate. De asemenea, sunt utilizate si mecanisme specifice interfetei ACPI.

Desi este o tehnologie promovata de Microsoft si companiile care produc echipamente prezentate ca fiind compatibile cu Plug-and-Play, mecanisme asemanatoare exista si in celelalte sisteme de operare disponibile pe piata. Kernel-ul Linux ofera un aceleasi facilitati ca si plug-and-play prin feature-ul de incarcare automata a modulelor de kernel disponibile in sistem, atunci cand un device este atasat computer-ului. In plus, sistemele bazate pe Linux permit incorporarea completa a driver-elor necesare in kernel prin recompilarea acestuia.

Totusi, sistemele Windows interactioneaza mai bine cu echipamentele atasate prin plug-and-play, datorita faptului ca acestea vin de cele mai multe ori cu drivere specifice, furnizate de dezvoltator, si nu ruleaza sub unui modul de kernel care ofera suport generic.

Avand in vedere cele mentionate, pe langa cei de la Microsoft un astfel de sistem se gaseste si in sisteme precum cele produse de Macintosh (firesc, avand in vedere ca MAC OS este derivat din FreeBSD) sau cele echipate cu distributii de Linux moderne (Ubuntu, Fedora, CentOS, etc.)

Evolutia Plug and Play

Plug-and-play a fost prima oara suportat de sistemul de operare Widows 95, dar de atunci a suferit schimbari importante o data cu specializarea sistemelor computationale si a cerintelor utilizatorilor.

Un punct de referinta a fost reprezentat de introducerea arhitecturii “OnNow” care cauta sa definesca o abordare globala a problemelor legate de configurarea sisitemului si a dispozitivelor cu care acesta interactioneaza. Un prim rezultat al acestei initiative a fost implementarea interfetei ACPI "Advanced Configuration and Power Interface" care defineste un nou sistem gestiune al resurselor, atat din punct de vedere al accesului si performantei, dar mai ales din punctul de vedere al consumului e energie . Prin aceste schimbari Plug and play-ul capata noi capabilitati cum ar fi: managementul consumului de curent si noi posibilitati de configurare, totate sub controlul total al sistemului de operare. Performante remarcabile au fost obtinute in domeniul notebook-urilor, remarcandu-se astfel o crestere a autonomiei dispozitivelor prin folosirea facilitatilor noi introduse.

Incepand cu Windows 2000 sistemul care pana atunci purta numele de Plug and Play se va numi **W**indows **D**river **M**model (WDM).

Noul sistem de operare Windows 8 aduce si mai multe imbunatatiri, introducand serviciul UpnP (Universal Plug and Play), care contine o colectie de drivere care permit controlarea unor dispozitive precum instalatii de aer conditionat controlabile peste IP.

Plug and Play în Linux

Sistemul de operare Linux nu a fost de la început un sistem plug and play, dar în prezent aceste probleme au fost rezolvate. Au fost adaugate mecanisme speciale pentru incarcarea si descarcarea automata a modulelor de kernel necesare controlarii unui dispozitiv, precum si tool-uri speciale prin care utilizatorul poate realiza depanarea si configurarea avansata a acestora. Fiecare driver realizează propria configurare a dispozitivelor folosind parametrii default sau parametrii transmisi de adiminstratorul de sistem.

Suportul pentru incarcarea dinamica a modulelor de kernel a permis dezvoltarea unei arhitecturi foarte flexibile si configurabile. Se obisnuieste ca atunci cand platforma hardware pentru care se construieste sistemul de operare este fixa si cunoscuta sa se realizeze o recompilare completa a kernel-ului, integrand direct in codul executabil al imaginii driverele necesare. Aceasta abordare aduce un plus de performanta si permite optimizarea dimensiunii imaginii. In cazul distributiilor generale se ofera un kernel "generic" care contine driverele necesare pornirii sistemului si incarcarii modulelor celorlalte module necesare la start-up.

In plus, distributiile de Linux au adaugat o serie de metode de control asupra resurelor amovibile. Distributiile mai vechi contineau HALd (Hardware Abstraction Layer Daemon) ca software pentru controlul dispozitivelor fizice atasate PC-ului. Din pacate, arhitectura acestuia implica o etapa de poll-ing la initializarea sistemului care incetinea initializarea. Alternativa la HALd o reprezinta UDEVd, un daemon mult mai flexibil si mai agil, care permite un start-up mai eficient. Ofera si facilitati avansate precum definirea de scripturi care pot fi executate la momentul identificarii unui dispozitiv nou sau la dezactivarea acestuia. De asemenea, se pot defini filtre pe baza carora unele clase de device-uri pot fi dezactivate, beneficiu extrem de important in implementara politicilor de securitate.

Figura 1. Functionarea udevd

Tehnologia plug-and-play în Linux se bazează în principal pe **Linux Device Model**, care necesita implementarea unor interfete standard definite in sursele kernel-ului si prin intermediul carora dispozitivele pot fi privite ca fisiere localizata in diverse directoare ale systemului (/proc, /sys, /dev, etc.).

Diferente fata de Linux

Ca si la Linux, in Windows (incepand cu Win 95) kernel-ul (sau portini componente ale acestuia) se ocupa de configurarea dispozitivelor si in acest sens este cu adevarat un sistem plug and play. Dispozitivele sunt descoperite automat in timpul secventei de boot sau la inserare (hotplug), determinand incarcarea automata a driver-elor corespunzatoare. De cele mai multe ori, driverele disponibile sub Windows sunt dezvoltate de catre producatorul echipamentului si ofera performante optime. La acest capitol, sistemele bazate pe Linux au de suferit. Cele mai multe module de kernel sunt dezvoltate ca module generice care pot controla o serie de componente bazate pe caracteristici (chipset-uri, arhitecturi, etc.) comune, ignorand astfel particularitatile fiecarei componente.

Tot datorita driverelor specializate, consumul de energie este mai bine optimizat in Windows. Arhitectura OnNow impune o serie de interfete stricte care permit controlarea setarilor dispozitivelor de tip plug-and-play intr-un detaliu mult mai fin. De cealalta parte, kernelul ofera o serie de guvernatori predefiniti (on-demand, performance,etc.) care configureaza dispozitiviele atasate fie prin interfata generica ACPI, fie prin atribute speciale definite la nivelul modulului de kernel folosit.

Ca si in cazul Linux-ului, in Windows (versiunile de dupa Windows 95) se aplica un algoritm de rezolvare a conflictelor ce apar la alocarea de resurse (ex. conflicte de IRQ) . In modelul anterior din Windows (legacy drivers), era necesara incarcarea explicita a driver-elor si initializarea dispozitivelor asociate acestuia la incarcare . Folosind plug and play, acest lucru nu mai este necesar, intrucat sistemul de operare se ocupa de aceste operatii (la detectarea unui dispozitiv se va apela o metoda speciala a driver-ului care va adauga dispozitivul).

In Windows (incepand cu Win2000), implementarea plug and play are mai multe componente software:

- managerul plug and play - are o componenta user-mode si una in kernel-mode si se ocupa cu detectarea si configurarea dispozitivelor fizice
- managerul de consum (power manager) - se ocupa cu managementul consumului (pentru a reduce consumul de energie al sistemului, anumite dispozitive pot fi trecute in stari speciale de economisire a energiei)
- registrii (registry) - contin o baza de date a componente lor hardware si software instalate in sistem si sunt folositi la identificarea si localizarea resurselor de catre dispozitive
- fisierele .inf (INF file)- descriu un dispozitiv, fiind necesar cate un astfel de fisier pentru fiecare dispozitiv la instalarea driver-ului; fiecare pereche dispozitiv/driver trebuie sa aiba un astfel de fisier
- drivere plug and play - desi exista drivere care folosesc doar partial arhitectura plug and play, se recomanda implementarea de drivere WDM (care respecta modelul Windows Driver Model) si care suporta complet arhitectura plug and play

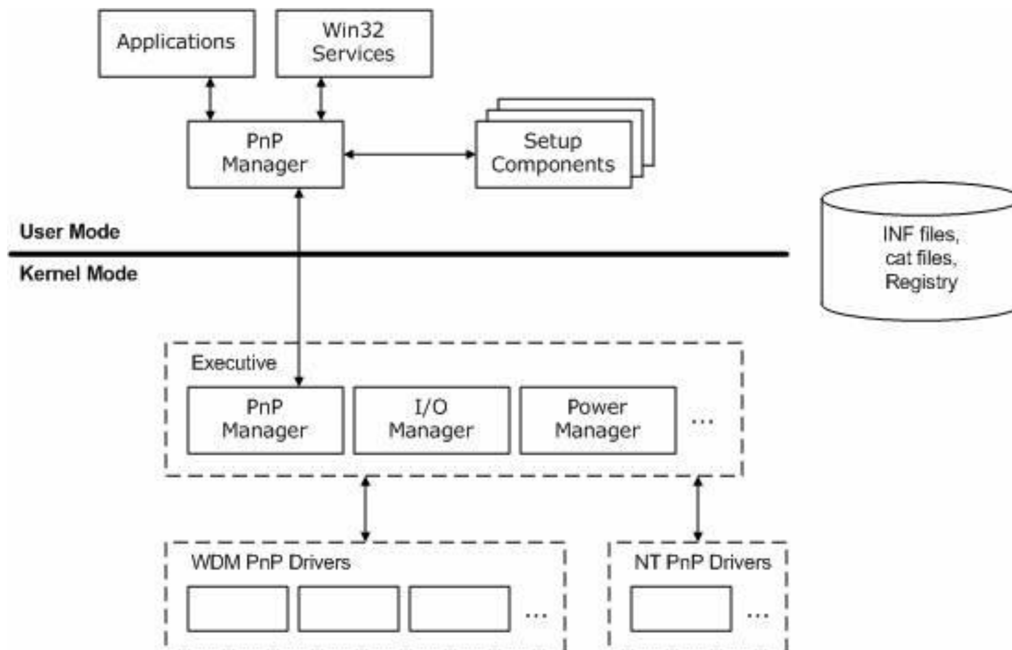

Figura 2. Componentele sistemului Plug and play (PnP)

Windows Driver Model (WDM)

Windows Driver Model este un model unificat, ce permite scrierea de drivere al căror cod sursă este compatibil pentru toate platformele Windiows. Un driver WDM (care respectă modelul Windows Driver Model) are următoarele caracteeristici:

- trebuie să aibă unul din tipurile de drivere WDM (bus driver, function driver, filter driver) şi să creeze dispozitive cu unul din tipurile WDM (Physical Device Object, Functional Device Object, Filter Device Object)
- trebuie să suporte plugg and play
- trebuie să suporte managementul consumului (power management)
- trebuie să suporte WMI (Windows Management Instrumentation); WMI este un mecanism prin care kernel-ul pune la dispozitia aplicaţiilor din user-mode informaţii (permite publicarea informaţiilor, configurarea dispozitivelor, un mecanism de notificări, logaarea evenimentelor, etc.)

Modelul WDM organizeaza driverele si dispozitivele intr-o stiva.

Astfel, driver-ele sunt impartite in trei categorii:

- bus drivers - drivere asociate magistralelor din sistem; este obligatoriu sa existe un astfel de driver pentru fiecare tip de magistrala din sistem; pot avea alte dispozitive conectate la magistrala; se afla la cel mai jos nivel in stiva de drivere
- function drivers - drivere pentru un dispozitiv individual; se afla deasupra driverelor pentru magistrala in stiva de drivere
- filter drivers - drivere care filtreaza cererile pentru un dispozitiv, o clasa de dispozitive sau o magistrala; se pot afla deasupra unui driver de magistrala (modifica in acest caz comportamentul dispozitivului) sau deasupra unui driver functional (adauga functionalitati suplimentare)

In stransa legatura cu tipurile de drivere, WDM defineste si tipul de obiecte ce descriu dispozitivele asociate fiecarui driver din stiva (Device_Object):

- Physical Device Object (PDO) - reprezinta un dispozitiv pe o magistrala pentru un driver dw magistrala; exista cate un astfel de obiect pentru fiecare tip de dispozitiv fizic si este responsabil cu controlul la nivel low-level al dispozitivului
- Functional Device Object (FDO) - reprezinta un dispozitiv pentru un driver functional; exista cate un astfel de obiect pentru fiecare functie logica sau abstracta care este o ferita nivelului superior
- Filter Device Object (filter DO) - reprezinta un dispozitiv pentru un driver de tip filtru; pot exista filtre atat pentru obiectele dispozitiv de tip fizic cat si pentru cele de tip functional

Functionarea unui driver plug and play si starile unui dispozitiv

Modelul WDM este o extensie a modelului anterior, NT. Astfel, DriverEntry ramane functia de initializare a drverului, numai ca dupa cum s-a precizat, nu se vor mai initializa dispozitivele asociate aici. Pentru aceasta, va exista o alta functie *AddDevice*, care va fi apelata de Plug and Play Manager pentru fiecare dispozitiv asaociat. Operatiile legate de dispozitiv sunt initiate de Plug and Play Manager prin transmiterea unui mesaj *IRP_MJ_PNP* (MajorFunction). Pentru a diferentia operatiile efectuate asupra dispozitivului se foloseste codul minor (MinorFunction). Acest cod poate avea una din urmatoarele valori:

- *IRP_MN_START_DEVICE* pentru initializarea sau reinitializarea dispozitivului cu resursele specificate
- *IRP_MN_QUERY_STOP_DEVICE* pentru a verifica daca dispozitivul poate fi oprit in vederea rebalansarii resurselor
- *IRP_MN_STOP_DEVICE* pentru a opri dispozitivul (pentru a fi repornit sau eliminat)
- *IRP_MN_CANCEL_STOP_DEVICE* pentru a informa ca nu se va opri dispozitivul, dupa o operatie *IRP_MN_QUERY_STOP_DEVICE*
- *IRP_MN_QUERY_REMOVE_DEVICE* pentru a verifica daca dispozitivul poate fi eliminat din sistem
- *IRP_MN_REMOVE_DEVICE* pentru a elimina dispozitivul din sistem (operatiile care deinitializeaza resursele initializate in functia *AddDevice*)
- *IRP_MN_CANCEL_REMOVE_DEVICE* pentru a informa ca nu se va elimina dispozitivul din sistem, dupa o operatie *IRP_MN_QUERY_REMOVE_DEVICE*
- *IRP_MN_SURPRISE_REMOVAL* pentru a informa ca dispozitivul a fost eliminat din sistem fara notificare in prealabil

Aceste coduri sunt valabile pentru toate driverele WDM. Pentru anumite tipuri de drivere (spre exemplu pentru driverele de tip magistrala sau pentru cele care au asociat un device de tip fizic si se ocupa cu managementul controlului la nivel low-level) sunt definite coduri suplimentare (spre exemplu, pentru aflarea capabilitatilor unui dispozitiv, pentru aflarea interfetei acestuia, aflarea resurselor conectate la o magistrala, etc.).

Dupa cum se poate observa din operatiile de mai sus, un dispozitiv trece prin diferite stari, in timp ce este configurat, pornit, eventual oprit pentru rebalansarea resurselor si posibil eliminat. Aceste stari se pot imparti in doua categorii: starile prin care dispozitivul trece atunci cand este adaugat in sistem si starile prin care trece dupa ce este adaugat.

In figura urmatoare sunt prezentate starile prin care trece un device in procesul de administrare bazat pe plug and play:

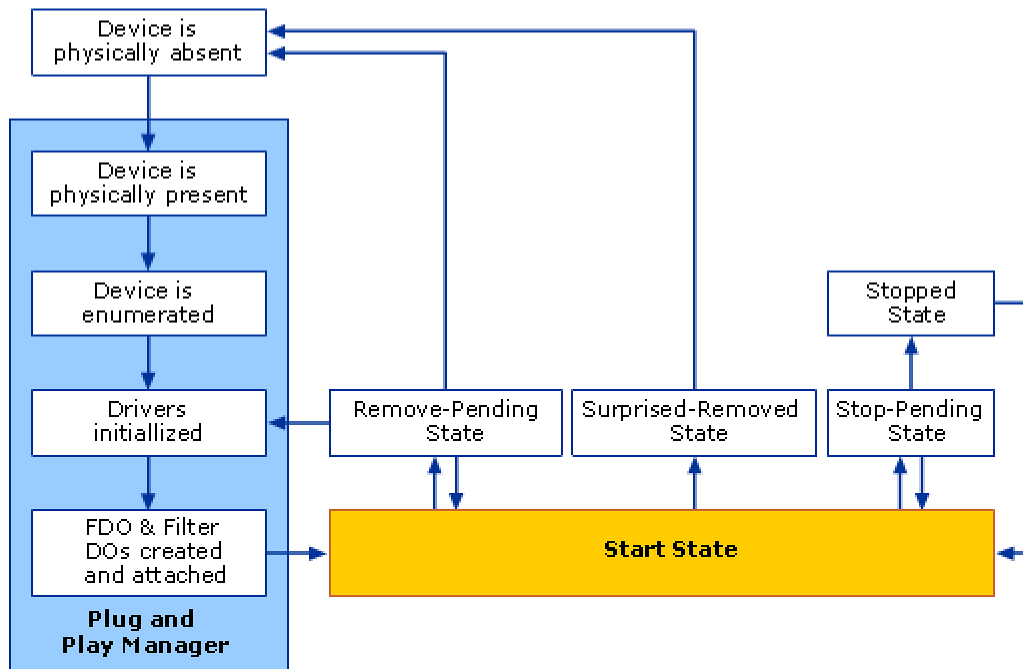

Figura 3. Stari Plug & Play

Initializarea driver-ului (*DriverEntry*)

La fel ca si in cazul modelului NT, driverele WDM se initializeaza in rutina *DriverEntry*. Spre deosebire de aceasta, initializeaza dispozitivele fizice (nu se mai apeleaza *IoCreateDevice*), ci doar se initializeaza functiile driver-ului.

Functiile ce trebuiesc initializate reprezinta functiile de dispatch pentru operatii de deschidere, scriere, citire, control, inchidere dispozitiv. Pe langa acestea, mai trebuie initializata functia pentru initializarea dispozitivelor (*AddDevice*) si functia pentru mesajele plug and play (*IRP_MJ_PNP*).

O functie *DriverEntry*pentru un driver plug and play va arata in modul urmator:

NTSTATUS DriverEntry(PDRIVER_OBJECT driver, PUNICODE_STRING registry)

Initializarea dispozitivului (AddDevice)

Dupa cum s-a observat mai sus, pentru initializarea dispozitivului exista o functie AddDevice, care va fi apelata de Plug and Play Manager in momentul descoperirii dispozitivului. Aceasta functie va prelua sarcina functiei DriverEntry din modelul NT si va initializa dispozitivul.

Prototipul acestei functii este urmatorul:

```
NTSTATUS AddDevice(
   IN PDRIVER_OBJECT  DriverObject,
   IN PDEVICE_OBJECT  PhysicalDeviceObject
);
```

, unde DriverObject este un pointer catre obiectul asociat driver-ului, iar PhysicalDeviceObject este un pointer catre dispozitivul fizic (PDO), creat de un driver de nivel mai jos.

La initializarea dispozitivului se creeaza o legatura interna pentru dispozitiv printr-un apel al functiei IoCreateDevice, se creeaza o legatura simbolica pentru userspace printr-un apel al functiei IoCreateSymbolicLink si se initializeaza datele private ale dispozitivului.

Pe langa aceste operatii, functia AddDevice mai trebuie sa adauge obiectul asociat driverului in stiva de obiecte. Aceasta operatie se realizeaza printr-un apel al functiei:

```
PDEVICE_OBJECT IoAttachDeviceToDeviceStack (
   IN PDEVICE_OBJECT  SourceDevice,
   IN PDEVICE_OBJECT  TargetDevice
);
```

, unde SourceDevice este un pointer catre obiectul asociat dispozitivului care apeleaza functia (noul varf al stivei dupa apel), iar TargetDevice este un pointer catre obiectul asociat altui dispozitiv (pointer catre PDO-ul stivei). Functia intoarce un pointer catre vechiul varf al stivei, deci un pointer catre dispozitivul situat sub dispozitivul apelant in stiva.

Intrucat obiectul a fost initializat in afara functiei DriverEntry, este necesara resetarea bit-ului pentru initializarea dispozitivului:

```
device->Flags &= ~DO_DEVICE_INITIALIZING;
```

Functie de dispatch pentru drivere WDM (IRP_MJ_PNP)

Dupa cum s-a observat, functia AddDevice doar initializeaza dispozitivul si datele sale private, fara a realiza operatii legate de dispozitivul fizic. Astfel, mai trebuie rezervat, initializat si configurat dispozitivul fizic. In acest scop, exista o noua functie de dispatch, IRP_MJ_PNP. Un IRP corepunzator este creat la initializarea dispozitivului, eliminarea sa sau cand se primesc

cereri din partea acestuia. Plug and Play Manager-ul va apela functia de dispatch corespunzatoare (inregistrata in DeviceEntry), care trebuie sa trateze aceste cazuri. Intrucat sunt mai multe operatii, diferentierea intre acestea se realizeaza prin codul minor al IRP-ului. Aceste coduri au fost prezentate mai sus, la functionarea unui driver plug and plaay si starile unui dispozitiv.

O astfel de functie de dispatch va diferentia intre aceste coduri si va avea urmatoarea structura:

NTSTATUS DispatchPnp(IN PDEVICE_OBJECT device, IN PIRP irp)

Dupa cum se poate observa, pentru fiecare din operatii se apeleaza o functie (ce va fi discutata in continuare), iar in cazul in care codul minor primit nu este suportat de driverul curent, este trimis urmatorului dispozitiv din stiva.

Transmiterea cererilor plug and play in stiva de dispozitive

Toate cererile plug and play sunt initiate de Plug and Play Manager si sunt transmise driver-ului care se afla in varful stivei de dispozitive. Indiferent ce coduri minore ale IRP-urilor sunt tratate de catre un driver, cele care nu sunt tratate de catre acesta trebuie trimise mai departe in stiva de dispozitive, la driverele de pe niveluri mai joase, care ar putea trata acele coduri. Aceasta operatie este necesara, intrucat un driver se bazeaza pe driverele de pe nivele mai joase pentru realizarea anumitor operatii (spre exemplu, un driver functional - FDO - se bazeaza pe driverul fizic - PDO). De asemenea, exista cereri care intereseaza toate driverele din stiva (cum ar fi informarea asupra opririi dispozitivului).

Pentru cererile care sunt tratate de driver, trebuie completata informatia IRP-ului legata de status (IoStatus) si apelata functia IoCompleteRequest. Pentru a transmite un o cerere plug and play in jos pe stiva, fara a astepta ca aceste pachete sa fie tratate de un driver, se apeleaza functia IoSkipCurrentIrpStackLocation, care elimina intrarea pentru stiva driver-ului curent din IRP si apoi IoCallDriver pentru a transmite IRP-ul driver-ului de la nivel inferior. Pointerul catre driverul de sub cel curent in stiva a fost obtinut in urma apelului IoAttachDeviceToDeviceStack de la initializarea dispozitivului ([[#Initializarea dispozitivului (AddDevice)| Initializarea dispozitivului (AddDevice)]]).

Functia PassDownPnP, care realizeaza aceste operatii si este apelata in functia de dispatch de mai sus are urmatoarea implementare:

NTSTATUS PassDownPnP(IN PDEVICE_OBJECT device, IN PIRP irp)

Pornirea dispozitivului (IRP_MN_START_DEVICE)

Plug-and-Play Manager-ul, la boot sau in momentul conectarii unui dispozitiv, le identifica si transmite un IRP cu codul minor IRP_MN_START_DEVICE driver-ului corespunzator. Pe masura ce identifica dispozitivele, Plug and Play Managerul le atribuie resursele cerute, cu evitarea pe cat posibil a conflictelor. Atunci cand transmite driver-ului IRP-ul, ii transmite si o lista cu resursele asociate dispozitivului fizic, in campurile Parameters. StartDevice.AllocatedResourcesTranslated si Parameters.StartDevice.AllocatedResources ale acestuia. Aceste campuri contin mai multe niveluri de vectori, si in final structura CM_PARTIAL_RESOURCE_DESCRIPTOR ce descrie resursele, care pot fi de patru tipuri: porturi, intreruperi, memorie si dma.

Resursele prezentate sunt de doua tipuri: raw (Parameters.StartDevice. AllocatedResources) si translated (Parameters. StartDevice.AllocatedResourcesTranslated). Resursele raw sunt cele intalnite in driverele din modelul NT si pentru care trebuiau realizate operatii de translatare. Din acest motiv, se vor folosi resursele translatate.

Oprirea dispozitivului (IRP_MN_STOP_DEVICE)

La primirea unui IRP cu codul minor IRP_MN_STOP_DEVICE, se vor executa operatii pentru oprirea dispozitivului. Aceste operatii sunt complementare celor executate in functia HandleStartDevice, la primirea unui IRP cu codul IRP_MN_START_DEVICE. Prin urmare, codul acestei functii este dependent de tipul de resurse detinute de dispozitiv. Functia HandleStopDevice, care realizeaza aceste operatii si este apelata in functia de dispatch de mai sus are urmatoarea implementare:

NTSTATUS HandleStopDevice(IN PDEVICE_OBJECT device, IN PIRP irp)

Eliminarea dispozitivului (IRP_MN_REMOVE_DEVICE)

La primirea unui IRP cu codul minor IRP_MN_REMOVE_DEVICE, se vor executa operatii pentru eliminarea dispozitivului din sistem. Aceste operatii sunt complementare celor executate in functia AddDevice si sunt aceleasi cu cele din DriverUnload, doar ca pentru un singur dispozitiv (cel dar ca parametru). Functia va deinitializa resursele, va sterge legaturile pentru dispozitiv si va transmite IRP-ul in jos pe stiva. In plus fata de operatiile cunoscute, apare deatasarea dispozitivului din stiva, printr-un apel al functiei IoDetachDevice.

Functia HandleRemoveDevice, care realizeaza aceste operatii si este apelata in functia de dispatch de mai sus are urmatoarea implementare:

NTSTATUS HandleRemoveDevice(IN PDEVICE_OBJECT device, IN PIRP irp)

Concluzii

Lucrarea de fata prezinta mecanismul Plug-and-Play implementat in sistemele de operare moderne Windows, dar si tehnologiile echivalente disponibile in universul Linux. Evolutia tehnologica din ultima vreme se indreapta inspre arhitecturi flexibile de kernel care sa permita integrarea facila a driverelor dezvoltate pentru perifericele disponibile pe piata. In Windows, de cele mai multe ori driverele sunt furnizate de producatorul componentei fizice si ofera performante mai bune decat modulele kernel generice folosite in kernelul de Linux.

Plug-and-Play a reprezentat un pas inainte intr-o lume in care din ce in ce mai multe dispozitive colaboreaza intre ele fara interventia utilizatorului, ajungand astfel din ce in ce mai aproape de ierarhia boards, pads si tabs prezentata de Mark Weiser in lucrarea „The Computer of the 21st Century".

BIBLIOGRAFIE:

1. http://tldp.org/HOWTO/Plug-and-Play-HOWTO.html
2. http://lwn.net/Articles/driver-porting/
3. http://msdn2.microsoft.com/en-us/library/ms798213.aspx
4. http://msdn2.microsoft.com/en-us/library/ms798213.aspx
5. The Windows 2000 Device Driver Book, Second Edition - Chapter 9. Hardware Initialization
6. Programming the Microsoft Windows Driver Model, Second Edition
7. Mark Weiser, „The Computer of the 21st Century"

Figura 1 http://daydreamer.idv.tw/rewrite.php/read-49.html

Figura 2 http://msdn2.microsoft.com/en-us/library/ms798233.aspx

Figura 3 http://technet.microsoft.com/en-us/library/cc781092(v=ws.10).aspx