DEVICE DRIVER DEVELOPMENT

FOR

CBK INFOTECH INDIA (P) LTD., BANGALORE.

PROJECT REPORT P-1231

Submitted in partial fulfillment of the requirements for the award of the degree

of

M.Sc Applied Science Software Engineering,

Of Bharathiar University,

Coimbatore.

Submitted By

KARTHIKEYAN.T

Reg. No. 0137S0034

Guided By

EXTERNAL GUIDE	**INTERNAL GUIDE**
Ms.V.HemaLatha, B.E,	Mrs. R.K.Kavitha, M.C.A, M.Phil,
System Designer.	Lecturer.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

KUMARAGURU COLLEGE OF TECHNOLOGY

COIMBATORE – 641 006

SEPTEMBER - 2004

KUMARAGURU COLLEGE OF TECHNOLOGY

(Affiliated to Bharathiar University)

Department of Computer science and Engineering

Coimbatore – 641 006

CERTIFICATE

This is to certify that the project work entitled

DEVICE DRIVER DEVELOPMENT

Done By

KARTHIKEYAN.T

Reg. No. 0137S0034

Submitted in partial fulfillment of the requirements for the award of the degree M.Sc Applied Science Software Engineering of Bharathiar University.

[illegible]
Professor and Head

[illegible] 24/9/04
Internal Guide

Submitted for the University examination held on ...29.09.04...........

[illegible]
Internal Examiner

[illegible]
External Examiner

CBK Infotech India (P) Ltd.

4/3, Sree Complex,
M.S.R.E.C. Road
Opp. NIAS, Mathikere,
Bangalore - 560 054
E-mail : cbk@satyam.net.in
Ph. : 3602402 / 3603995

Ref No. CBK/CT/2009/2004/03 Date:20.09.2004

TO WHOMSOEVER IT MAY CONCERN

This is to certify that **Mr. T. Karthikeyan, (Reg . No. 0137S0034)** of Kumaraguru College Of Technology, Coimbatore (affiliated to **Bharathiar University**), has successfully completed the project titled **'Device Driver Development'** in compliance with the requirement of partial fulfillment of the **Master Of Science (Software Engineering).** He was associated with us during the period from June -2004 to September-2004.

As per our company policy the source code is a property of CBK Infotech India (P) Ltd. And it cannot be disclosed.

For and on behalf of CBK Infotech India Pvt Ltd.

CBK INFOTECH INDIA LTD

20/9

HR Manager

DECLARATION

I hereby declare that the project entitled "**Devices Driver Development** ", for **CBK InfoTech India(P) Ltd**. Bangalore submitted to **Kumaraguru College of Technology**, Coimbatore, affiliated to **Bharathiar University** as the project work of **M.Sc Applied Science Software Engineering,** is a record of original work done by me under the supervision and guidance of Ms. V.HemaLatha ,B.E and Mr. K.R.Baskaran, B.E, M.S and this project work has not found the basis for the award of any Degree/Diploma/Associate ship/Fellowship or similar title to any candidate of any university.

Place: Coimbatore

Date: 24.09.04

T. Karthikeyan

Signature of the Student

ACKNOWLEDGEMENT

To add meaning to the perception, it is my indebtedness to honor a few who had helped me in this endeavor, by placing them on record.

With profound gratitude, I am extremely thankful to Dr.K.K.PADMANABAN B.Sc. (Engg), M.Tech, Ph.D., Principal, Kumaraguru College of Technology, Coimbatore for providing me an opportunity to undergo the M.Sc APPLIED SCIENCE SOFTWARE ENGINEERING course and thereby this project work also.

I extend my heartfelt thanks to my Computer Science & Engineering Department head, Prof.Dr.S.THANGASAMY B.E (Hons), Ph.D., for his kind advice and encouragement to complete this project successfully.

It's my privilege to express my deep sense of gratitude and profound thanks to Ms.V.HemaLatha, B.E System Designer, CBK InfoTech India (P) Ltd., Bangalore for having allowed me to do my project work in her esteemed team and for helping me in all means in successful completion of this project work.

Gratitude will find least meaning without thanking my Project coordinator Mr.K.R.BASKARAN, B.E, M.S., Assistant Professor, Dept of Information Technology and my guide Mrs.R.K.Kavitha, M.C.A, M.Phil Lecturer, Dept of Computer Science & Engineering for the valuable guidance and support throughout my project.

My gratitude is due to all staff members of CSE department, my parents and all my friends for their moral support and encouragement for successful completion of my project.

KARTHIKEYAN.T
(Reg.No:0137S0034)

SYNOPSIS

The aim of this project is to provide a device driver for Microsoft Windows NT that will allow touching any devices connected with Peripheral Component Interconnect (PCI) system bus. The device driver is to provide the full-required functionality of a memory, I/O and Interrupt operation on any PCI device, which is connected to PCI bus. Driver can touch entire memory location of any device that is connected with PCI bus. Driver supports memory mapped I/O and I/O mapped I/O operations. Driver search's for the selected device and communicates with the particular device. Driver runs at kernel level of the operating system.

Application gives complete details of devices connected to PCI bus. It gives an opportunity to select devices and get complete information about devices. You can select any device for I/O, memory or Interrupt operation. Application will pass message to driver with complete information.

Specifically, the device driver and the user library are to be implemented in Microsoft Visual C++ using the Microsoft Windows NT Device Driver Kit (DDK) and Windows NT Software Development Kit (SDK) environment.

According to the Windows NT terminology the device driver developed in this project is a kernel-mode driver (Microsoft Corporation 1996). The device driver is designed to operate only with PCI interface devices. Alternative system buses, such as ISA, are not supported. The driver provides an interrupt service routine for handling hardware interrupts. The driver also provides routines for handling read () and write () system calls. The C library developed in this project provides the necessary interface between the user processes and the Driver.

TABLE OF CONTENTS PAGE.NO

1.0 INTRODUCTION

1.1 PROJECT OVERVIEW

The goal of this project is to provide a device driver for Microsoft Windows NT that will allow touching any devices connected with Peripheral Component Interconnect (PCI) system bus. The device driver is to provide the full-required functionality of a memory, I/O and Interrupt operation on any PCI device, which is connected to PCI bus. Driver can touch entire memory location of any devices connected with PCI bus. Driver support memory mapped I/O and I/O mapped I/O.

Driver will search any selected device and communicate with that device. Driver runs at kernel level of operating system. So the Kernel level driver gives an opportunity to the user level application to communicate with the driver. Driver receives all messages, which comes from the user level application. According to the user's message, driver performs the read/write operation. If there is any problem with driver or device then the driver should not start and flash proper message. At the time of unloading, the driver should not get crashed.

The user application gives complete details of the devices connected to PCI bus. It gives an opportunity to select one device at a time and get complete information about devices. You can select any device for I/O, memory or Interrupt operation. Application will pass message to the driver with complete information. Driver runs at ring 0 level and application runs at ring 3 level. So when the application wants to communicate, the driver control should jump from ring 3 level to ring 0 level.

Specifically, the device driver and the user library are to be implemented in Microsoft Visual C++ using the Microsoft Windows NT Device Driver Kit (DDK) and Windows NT Software Development Kit (SDK) environment. The implementation of the device driver will be tested in the working environment of PC-compatible platforms.

Applications that are based on the device driver and the C user library, will be tested with one or more PCI devices.

According to the Windows NT terminology, the device driver developed in this project is a kernel-mode driver (Microsoft Corporation 1996). The device driver is designed to operate only with PCI interface devices. Alternative system buses, such as ISA, are not supported. The driver provides an interrupt service routine for handling hardware interrupts. The driver also provides routines for handling read () and write () system calls. Therefore, the application is capable of using the operating system's services for transferring data to/from the device. The C library developed in this project provides the necessary interface between the user processes and the driver. Practical issues concerning the communication performance justify this functionality of the device driver and the user library.

1.2 ORGANIZATION PROFILE

CBK InfoTech: Partners of TCS (Tata consultancy Services) is a world class software led IT services. CBK InfoTech is a IT service company providing a range of value added software services to:

- Hardware product companies
- Software product companies
- End-user in large and medium business organization.

ABOUT CBK INFOTECH INDIA (P) LTD.

The information is the hallmark of today's world. A driver for productivity and the ability to offer quality solutions on information super high way are the key to development of CBK InfoTech India (P) Ltd.

The essence of true development since 1988 by enhancing growth with the presence of social justice. In promoting and cherishing the growth of those associated with clients who are the true partners in progress.

There is no shortcut to success so as in the case of IT industry too. It is never possible without innovation, an eye for vision, a strong will to succeed and unlimited quality service. Quality objectives, precise and time bound are the root criteria for success and development is not an exception with CBK.

CBK will leave no stone unturned to reach its customer to the topmost rung of ladder success. A result that is translated at CBK, i.e. - in tune with technology with time and trust, truth and tradition, and requirement is the principle assets. CBK has two divisions working at the moment - Training division as Compu Home and a software development division. It is the development division that is offering this project training as detailed in this document.

CBK provides the state of the art technology like COM, COM+, Active-X, ASP, 3-tier solutions etc. and limited support of its clients in India and abroad.

CBK also provides Consultancy services for all IT related matters to its clients. With the revolving strategy and re-structuring, CBK has now started offering Web based solutions and gearing towards providing the E-Commerce / M-Commerce solutions to its existing and new clients.

TRAINING TIE-UP

CBK INFOTECH having status of ATC of TATA CONSULTANCY SERVICES for committed to excellence in corporate training. CBK have the unique advantage of combining a management perspective with in-depth technical knowledge in all our training solutions. CBK offer a wide range of training programs to meet the requirements of corporate clients.

OTHER TIE-UPS

The following are the measure Tie-ups of CBK INFOTECH in the areas of Software Development, Consultancy and Training.

1. Project Development Partner of TCS
2. Technology Team Development of RGSL
3. R&D and technology Team Development of APSON.com (p) LTD.
4. Prototype Development for VFM Software Solutions (p) LTD.

And Many More........

WORK RELATED AREAS @ CBK:

1. Developing Device And Device Drivers
2. Web enabled applications development
3. Client / Server Applications Development
4. Embedded System
5. Research & Development in WAP and WEB related conversing technologies
6. Corporate training
7. High-end User Training (Vocational)
8. Industrial Automation
9. Data Processing
10. One-Wire and Tiny Technology
11. Palm top/Hand Held PC Application Development/ E-CRM……

It is the policy of CBK to design, develop, deliver, maintain and support high quality software solutions. This is done not only to meet the client's requirements but also to exceed their expectations by being their true partners to the ladder of success. CBK extend its services to its clients by providing skilled manpower resources on contractual basis. This leads to a dedicated human resources development program.

TECHNICAL

The technical team at CBK has a combined IT experienced of more than 15 years in tools as follows

- C / C ++ / VB / VC++ / Java / Developer 2000
- Access / MS SQL Server / Oracle
- HTML / DHTML / ASP
- COM / COM+ / D-COM / MTS
- Visual Interdev / Front Page / Hotdog
- Adobe products / Macromedia Products
- Etc…

2.0 SYSTEM STUDY AND ANALYSIS

2.1 SOFTWARE REQUIREMENT SPECIFICATION

The design of device drivers is influenced mainly by three factors:

- The hardware architecture of the interface adapters
- The system bus of the host computer
- Operating system upon which the driver is supposed to execute.

The aim of the project is to detect the existence of the PCI buses in the system. If the bus is present then identifying the PCI devices and capture the resources and complete Information about the devices present, to perform read or write operation. This operation is performed on Input /Output Port or Memory of the PCI device. Also information about the interrupt present is read.

TASK ACCOMPLISHMENT:

The task is accomplished by writing driver program with the help of SDK and DDK for Windows NT Operating System.

SOFTWARE REQUIREMNTS:

a) DEVICE DRIVER

- WinNT Driver Development Kit (DDK)
- Software Development Kit (SDK)
- Language 'C'
- OS – Windows NT 4.0

b) APPLICATION

- VC++ (MFC)
- OS – Windows NT 4.0

2.2 EXISTING SYSTEM

- In the existing system separate drivers are present for the devices connected to the PCI-BUS.
- There is no separate driver that deals with all the devices connected to the PCI-BUS.
- For example if the devices like floppy drive, CD-ROM, or Mouse are present in the system connected to the PCI-BUS then there a separate driver is required to operate these devices.
- These separate drivers cannot be linked with the other device drivers.
- To over come this problem the proposed system has been developed

2.3 PROPOSED SYSTEM

A device driver for Microsoft Windows NT is designed, that will allow touching any devices connected with Peripheral Component Interconnect (PCI) system bus. The device driver provides the full-required functionality of a memory, I/O and Interrupt operation on any PCI device, which is connected to PCI bus. Driver can touch entire memory location of any device that is connected with PCI bus.

Driver support memory mapped I/O and I/O mapped I/O operations. Driver searches for the selected device and communicates with the particular device.

The user application gives complete details of the devices connected to PCI bus. It gives an opportunity to select one device at a time and gets complete information about devices. You can select any device for I/O, memory or Interrupt operation. Application will pass messages to the driver with complete information

The device driver is designed to operate only with PCI interface devices. Alternative system buses, such as ISA, are not supported. The driver provides an interrupt service routine for handling hardware interrupts.

The driver also provides routines for handling read () and write () system calls. Therefore, the applications opening the device can use the operating system's services for transferring data to/from the device.

In the existing system there is a separate driver for all the devices connected to the PCI-bus. This problem has been over come by developing a single driver for all the devices connected to the PCI-bus. This operation is done by writing a single driver entry to the driver.

For example consider the devices like floppy drive, Mouse, CD-ROM that is present in the system connected to the PCI-bus. There will be a single driver that can perform the process of identifying the devices and to get the details regarding the devices and the port or memory address that is used for developing drivers of the separate devices.

3.0 PROGRAMING ENVIRONMENT

3.1 HARDWARE CONFIGURATION

Machine	:	Intel 80x
Processor	:	Intel Pentium-III
Clock speed	:	450Mhz
Floppy disk	:	1.44 MB
Ram	:	16MB RAM.

- 60MB hard-disk space for a minimum DDK installation; 95MB for a full installation. (The same space is required for all platforms; a manual installation is equivalent to a full installation.)
- 80MB hard-disk space for a full installation of the HCT test suites. (See testing Your Driver for more details on system requirements and installing various testing configurations.)
- A second machine capable of running Windows NT for kernel debugging.
- Any PCI Device

3.2 DESCRIPTION OF SOFTWARE AND TOOLS USED

➢ **Microsoft Visual C++**

Visual VC++ 6.0 is designed to deploy applications across the enterprise and to scale to nearly any size needed. The ability to object models, database integration, server components, interacting with operating

System and Internet/intranet applications provide an extensive range of capabilities and tools for the developer.

Why Visual VC++?

Among the modern programming languages Visual VC++ plays a pivotal role. In the field of development it can apply in to multiple purposes very easily. Moreover it is a product of Microsoft. As like any other product of Microsoft Visual VC++ is also very easy to use. We can easily develop applications and enhance it using Visual VC++. Visual VC++ ranges from lightweight Visual VC++ Script programming, to application- specific programming with Visual VC++ for Applications, and finally, full-fledged enterprise development.

- Visual Platform
- Ease of Use
- Flexibility
- Ease of Enhancement
- Easy to Understand

Windows has been one of the most popular graphical user interface (GUI) operating systems, and Microsoft revolutionized Windows development with the advent of Visual VC++. Windows works in an event-driven environment, meaning the user is in control, and programs need to create and respond to events (such as a mouse click). Visual VC++ became one of the first "Visual" tools to provide an elegant interface for working in this environment. And the sixth iteration adds the ability to work in a different type of visual environment-the Internet. Visual VC++ has been popular for one other kind of development-Databases. Visual VC++ 6.0 gives us even more tools in the IDE to work with databases.

Microsoft has added significant functionality in several core areas. They are outlined as follows:

- DATA ACCESS A key addition for helping programmers build database applications is support for ActiveX Data Objects, Microsoft's new data access technology. Development tools include a SQL editor, data environment for developing data objects, and data report designer.
- INTERNET FEATURES: What may be the most significant new functionality is the ability to create both server-and client-side Internet applications. Visual VC++ finally comes to the Internet.
- COMPONENT CREATION Visual VC++ continues to emerge as a world-class development tool for creating and working with components ranging from ActiveX controls to integration with Microsoft's Transaction Server.
- NEW AND UPDATE CONTROLS: Microsoft continues its legacy of creating and enhancing controls for supporting rapid application development.
- LANGUAGE FEATURES: True to the long tradition of Visual VC++, there have been many language enhancements that make working with strings, numbers, and objects even easier and more powerful.
- WIZARDS: Wizards and more wizards have been added to the tool set to support rapid application development of enterprise-level applications.

➢ Microsoft Windows NT 4.0

Windows NT (Windows New Technology) is a 32-bit secure operating system (OS) that uses a Graphical User Interface (GUI) for graphical, interactive user control. It can support multiple processors (up to 16 in Windows NT Server editions) it is a preemptive and multi-tasking OS. Rather than the traditional method of applications allocating central processing unit (CPU) time, the OS makes these assignments, preventing most applications from hanging the operating system, resulting in few, if any, operating system crashes. In previous versions of Windows, Windows has been an add-on to the Disk Operating System (DOS). WindowsNT is a complete operating system by itself.

Second, its developers gave NT a multitasking preemptive-scheduling system. Neither DOS nor Windows 3.x is capable of true multitasking with preemptive scheduling. Without add-on software, DOS can execute only one program, or task, at a time. Windows 3.x can execute several programs concurrently, but they must be well behaved; that is, each program must be aware that other programs may need to run, and it must therefore yield the machine at regular intervals. This design means that a buggy or malicious program can halt the computer simply by entering an infinite loop in which it never yields. In NT, a centralized scheduling authority doles out CPU time to programs that need it. Once a program's turn has ended, the scheduler has the power to preempt it and give another program a turn.

- COMPACTIBILITY: The OS should support a wide range of existing s/w & h/w.
- ROBUSTNESS AND RELIABILITY: The OS has to resist the attacks of naïve or malicious user & individual applications should be as isolated from each other as possible.

- o PORTABILITY: The OS should be able to run on a wide variety of current and future platforms.
- o Extensibility: It should be possible to add new features and support new I/O devices without disturbing the existing code base.
- o Performance: The OS should be able to give reasonable performance on commonly available h/w. It should also be able to take advantage of features like multiprocessing hardware.

♦ CHARECTERISTICS OF WINDOWS NT

- MULTITHREADING

The unit of execution and scheduling on Windows NT is the thread. An initial thread is created when a process is created. That thread may create additional threads. In Windows NT, each thread has its own scheduling priority and is autonomous in terms of scheduling i.e. the OS does not take into account the process to which a thread belongs when it makes scheduling decisions.

- MULTITASKING

Windows NT allows multiple units of execution to run simultaneously. It rapidly switches among these units of execution, allowing each to run for a short period of time. This characteristic is termed as multithreading. In NT, multiple threads may run at one time. The decision as to which thread is selected is almost entirely based on priority. Nt has 32 possible thread priorities.

Priorities 0-15 are Dynamic priorities
Priorities 16-31 are Real-Time priorities

NT implements what is known as Pre-emptive multitasking. This is based on time slicing. Neither DOS nor Windows 3.x is capable of true multitasking with preemptive scheduling. Without add-on software, DOS can execute only one program, or task, at a time. Windows 3.x can execute several programs concurrently, but they must be well behaved; that is, each program must be aware that other programs may need to run, and it must therefore yield the machine at regular intervals. This design means that a buggy or malicious program can halt the computer simply by entering an infinite loop in which it never yields. In NT, a centralized scheduling authority doles out CPU time to programs that need it. Once a program's turn has ended, the scheduler has the power to preempt it and give another program a turn.

- MULTIPROCESSING

NT supports only symmetric multiprocessing (SMP). Here all systems share the same main memory and each system has equal access to the peripheral devices. The OS runs on all the processes in the SMP system. In NT there is no concept of master and slave CPUs.

The Windows NT architecture supports SMP systems with up to 32 CPUs. By default, the Win NT WORKSTATION can support 2 processors, Windows NT Server Systems can support 4 processors and Windows NT Enterprise Edition can support up to 8 processors.

- INTEGRATED NETWORKING

Windows NT has always supported multiprotocol networks. A few protocol families which NT supports are AppleTalk, TCP/IP, DLC etc.

- DEMAND PAGED VIRTUAL MEMORY

Windows NT utilizes a virtual memory architecture in which each process has its own 4GB virtual address space. This virtual address space is subdivided into pages. Each page is of 4KB in X-86 architecture. Typically user applications have access to 2GB of their process's virtual address space, with the remaining 2GB of address space to be used by the system. The NT virtual memory model allows the same physical addresses space to appear within the virtual address space of multiple processes

- USER VS KERNAL MODE

The NT architecture is divided into two modes –the user mode & the kernel mode. User mode is the least-privileged mode NT supports; it has no direct access to hardware and only restricted access to memory i.e. they can't touch parts of memory that are not specifically assigned to them. Kernel mode is a privileged mode. Those parts of NT that execute in kernel mode, such as device drivers and subsystems such as the Virtual Memory Manager, have direct access to all hardware and memory.

Other operating systems, including Windows 3.1 and UNIX, also use privileged and no privileged modes. What makes NT unique is where it draws the line between the two.

- MICRO KERNAL ARCHICTURE

NT is sometimes referred to as a micro kernel-based operating system. The idea behind the pure microkernel concept is that all operating system components except a small core (the microkernel) execute as user-mode processes. But the core components in the microkernel execute in privileged mode, so they access hardware directly

Microkernel architecture gives a system configurability and fault tolerance. Because an operating system subsystem like the Virtual Memory Manager runs as a distinct program in microkernel design, a different implementation that exports the same interface can replace it. If the Virtual Memory Manager fails in a microkernel design, the operating system can restart it with minimal effect on the rest of the system. But in a monolithic operating system design (e.g., DOS and Windows 3.1), the entire operating system must be rebuilt to change any subsystem. I.e. if the Virtual Memory Manager has a bug in a monolithic system, the bug is likely to bring down the machine.

A disadvantage to pure microkernel design is slow performance. Every interaction between operating system components in microkernel design requires an inter process message, which results in context switches

NT takes a unique approach, known as modified microkernel that falls between pure microkernel and monolithic design. In NT's modified microkernel design, operating system environments execute in user mode as discrete processes, including DOS, Win16, Win32, OS/2, and POSIX .The basic operating system subsystems, including the Process Manager and the Virtual Memory Manager, execute in kernel mode, and they are compiled into one file image. These kernel-mode subsystems are not separate processes, and they can communicate with one another by using function calls for maximum performance.

Thread scheduling and thread dispatching are the responsibilities of the microkernel. NT's Executive components use basic hardware functionality implemented in the microkernel. The microkernel, which is known in NT as the Kernel, contains the scheduler. The Kernel also manages the Executive's use of NT's hardware and software interrupt handlers and exports synchronization primitives

- MULTIPLR OS EMULATION

NT supports execution of Win32, POSIX, OS/2, DOS & Windows 3.1 programs with their native semantics. NT's user-mode operating system environments implement separate operating system APIs. The degree of NT support for each environment varies. NT's operating system environments rely on services that the kernel mode exports to carry out tasks that they can't carry out in user mode. The services invoked in kernel mode are known as NT's native API This API is made up of about 250 functions that NT's operating systems access through software-exception system calls.

Native API requests are executed by functions in kernel mode, known as system services. To carry out work, system services call on functions in one or more components of NT's Executive.

Device drivers are dynamically added NT components that work closely with the I/O Manager to connect NT to specific hardware devices, such as disks and input devices.

- PROCESSOR ARCHICTURE INDEPENDENCE

Windows NT was designed to work on a wide variety of processors. To facilitate this, most Windows NT code is written in the C programming language. Use of assembly language has been kept to a minimum.

Device drivers and the Kernel use the HAL to interact with the computer's hardware. The HAL exports its own API, which translates abstract data into processor-specific commands. NT is portable across processor types because processor-specific code is restricted to the Kernel and the HAL. This situation means that when NT is ported to a new processor, only the Kernel and the HAL must be converted. The rest of NT's code is written in C and C++ and can simply be recompiled for the new processor.

- OPERATING SYSTEM ENVIRONMENTS

NT's operating system environments are implemented as client/server systems. As part of the compile process, applications are bound by a link-time binding to an operating system API that NT's operating system environments export. The link-time binding connects the application to the environment's client-side DLLs, which accomplish the exporting of the API. For example, a Win32 program is a client of the Win32 operating system environment server, so it is linked to Win32's client-side DLLs, including Kernel32.dll, gdi32.dll, and user32.dll.

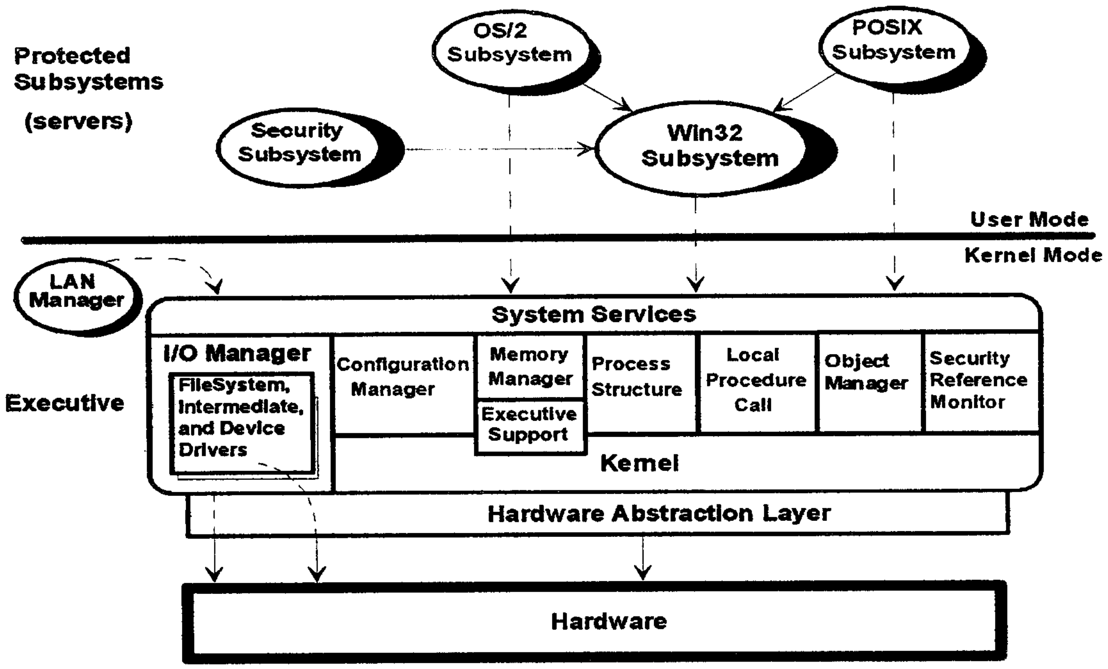
Client-side DLLs carry out tasks on behalf of their servers, but they execute as part of a client process. In some cases a client-side DLL can fully implement an API without having to call upon the help of the server ex. Read File, in other cases, the server must help out ex Create Process.

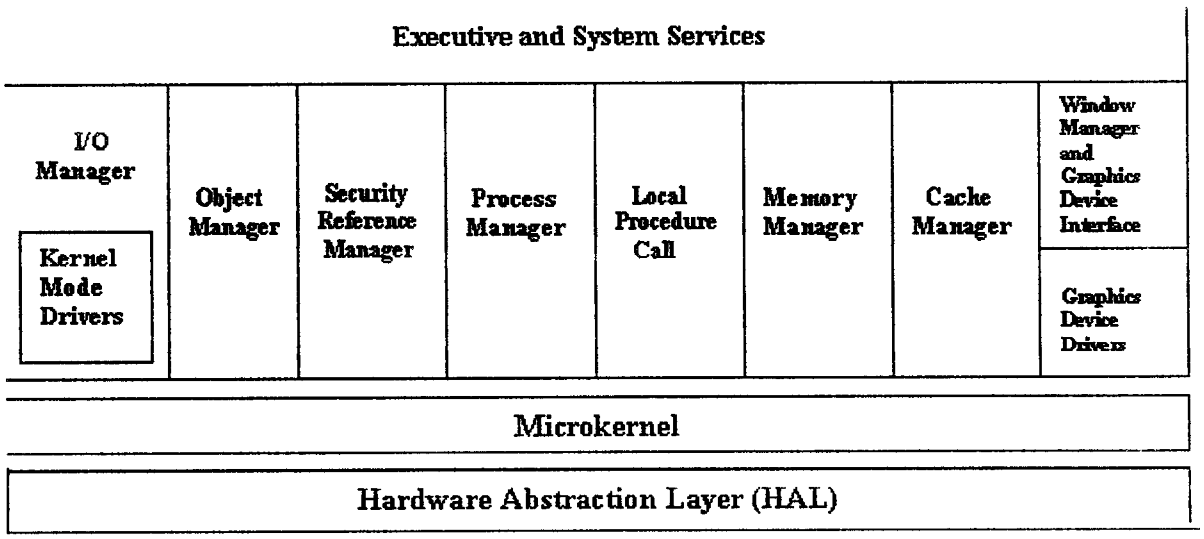
User components, In NT 3.51, whenever a Win32 program makes a drawing or user-interface call, the GDI or User client-side DLLs make LPC calls to the Win32 server

The server's aid is usually necessary only when global information related to the environment must be updated. When the client-side DLL requires help from the server, the DLL sends a message known as a local procedure call (LPC) to the server. When the server completes the specified request and returns an answer, the DLL can complete the function and return control to the client. Both the client-side DLL and the server may use NT's native API when necessary.

However, Microsoft has removed the most LPC-intensive portion of NT 3.51's Win32 operating system environment. The Win32 environment includes graphics and user-interface functions, which are implemented in its graphics device interface (GDI) and (CSRSS.EXE). Those LPC calls to the server cause Win32's sluggish performance.

But in NT 4.0, the User and GDI components have been moved from user mode into kernel mode as a new Executive subsystem, Win32K.SYS. When a drawing call is made, the client-side GDI's DLL makes a new native system call into kernel mode, where the request is carried out (Win32 native system calls didn't exist in NT 3.5x). There is no message passing and no context switches--just a switch from user mode to kernel mode and back. This optimization has a dramatic effect on the performance of Win32 applications


Protected Subsystems (servers)
OS/2 Subsystem
POSIX Subsystem
Security Subsystem
Win32 Subsystem
User Mode
Kernel Mode
LAN Manager
System Services
Executive
I/O Manager
FileSystem, Intermediate, and Device Drivers
Configuration Manager
Memory Manager
Executive Support
Process Structure
Local Procedure Call
Object Manager
Security Reference Monitor
Kernel
Hardware Abstraction Layer
Hardware


Executive and System Services
I/O Manager
Kernel Mode Drivers
Object Manager
Security Reference Manager
Process Manager
Local Procedure Call
Memory Manager
Cache Manager
Window Manager and Graphics Device Interface
Graphics Device Drivers
Microkernel
Hardware Abstraction Layer (HAL)

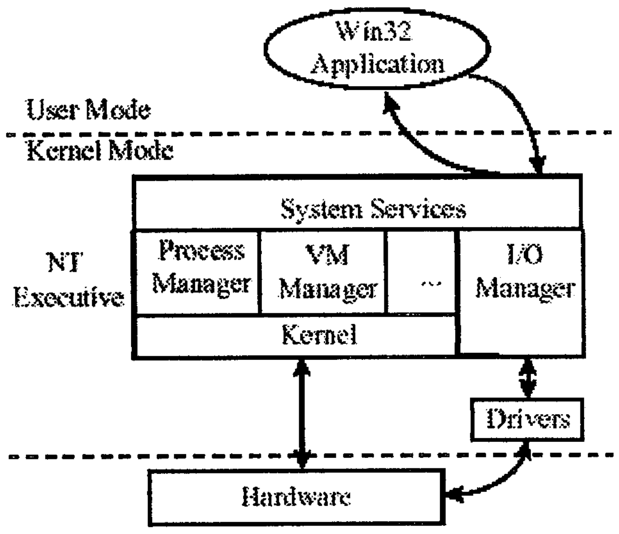


Figure 4. Windows NT input-output system (adapted from Custer 1993)

♦ THE WINDOWS NT 4.0 BOOT PROCESS

First of all, there are a handful of files required for a successful boot:

NTLDR - The operating system loader, which must reside in the root of the boot drive. In a multi-boot environment, it will be used to start the other systems initially. This file is hidden, system, and read-only.

BOOT.INI - a text file, which is used to build the OS Selection menu, and gives the path to each OS available. This file also must be in the root of the boot drive, and is read-only, system.

BOOTSECT.DOS - This file contains the boot sector of the Operating System that was on the hard drive previous to installing NT. NTLDR will use this to boot in a multi-boot environment if an OS other than NT was chosen from the boot menu. It too, is a hidden system file that must be in the root of the boot drive.

NTDETECT.COM - This program examines the hardware on the machine, and builds a list, which it passes back to NTLDR to be used to build the Hardware Hive of HKEY_LOCAL_MACHINE in the Registry. This file is also a hidden, read-only system file, in the root of the boot partition.

NTOSKERNEL.EXE - The kernel of the OS itself, which resides in the WINNT\SYSTEM32 directory.

NTBOOTDD.SYS - This device driver file will only be used on systems that boot from a SCSI disk on which the SCSI adapter BIOS is disabled.

SYSTEM - This file is located in the WINNT\ SYSTEM32\CONFI`G folder, and controls which drivers and services are loaded during the Windows NT startup.
Next when the machine is first powered on, it will go through a series of steps before NT actually begins booting:

POST - Power on Self Test (usually counts memory, offers an opportunity to enter the BIOS settings, etc.).
The machine locates the boot device, and loads the MBR (Master Boot Record) into memory, which runs the program, which is in it.
The MBR's program will locate the active partition and load the boot sector into memory from it.
From the boot sector, NTLDR will be loaded into memory and run.

This brings us to the actual NT booting process.

➢ NTLDR switches the processor to a 32-bit flat memory model, supporting up to 4 GB of RAM (physically installed).

➢ NTLDR starts what is called a mini file system. Windows NT can read one of three file formats: FAT, NTFS & CDFS.

➢ NTLDR reads the BOOT.INI file, and displays the operating system selections in the Boot Menu. If Windows NT is selected, NTLDR will run NTDETECT.COM. If another OS is selected, NTLDR will load and run BOOTSECT.DOS, and pass control to it, and exit. The other OS will continue as though the machine had just booted. If BOOTSECT.DOS is missing or corrupt, it must be replaced or reconstructed in order to boot to the other OS.

➢ NTDETECT.COM scans the machine's hardware (you will notice the keyboard lights and modem lights flash at this point, as it scans the various ports). The information gathered during this phase will be passed back to NTLDR.

➢ NTLDR then loads NTOSKRNL.EXE and passes the hardware information to it. This technically ends the "boot phase" and begins the "load phase."

There are four more phases to go before NT is officially up and running. They are the Kernel Load, Kernel Initialization, Services Load, and Subsystem Start Phases.

➢ During the Kernel Load phase, the HAL (hardware abstraction layer) is loaded, which hides the physical hardware from applications.
The system hive of the registry is loaded next, and scanned for drivers and services that should be loaded.

➢ During the Kernel Initialization phase, the screen is blue. The drivers are initialized and loaded, and the registry's CurrentControlSet is then saved, and the Clone control set is created, but not saved. The registry hardware list is then created from the information gathered earlier.

➢ In the Services Load phase, the session manager is started (SMSS.EXE), which runs any programs listed in HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\ Control\Session Manager: Boot Execute. After this, the session manager sets up the page file(s). Next the Clone Control Set is written to the registry. The last thing the session manager does is loading the required subsystems (by default, only Win32).

➢ In the Subsystem Start phase, WINLOGON.EXE is automatically started, which starts the Local Security Authority (LSASS.EXE) and brings up the logon dialog box (CTRL+ALT+DEL). The Service Controller (SCREG.EXE) is then run, which looks through the registry for services that are set to automatic load, and loads them.

➢ The last part of a Windows NT boot is the user logon. Once the user logs on successfully, a boot is considered complete, and the Clone control set is copied to the Last Known Good control set.

➢ **Microsoft Windows NT 4.0 DDK_(Device Development Kit)**

- Windows NT DDK drivers are built using the build utility, which uses a set of rules and project files that specify how drivers should be created.
- It is a product of Microsoft Corporation used to compile the driver file.
- It compiles the driver coding into a system file.

➢ **Debug Viewer**

This software is used to debug and test the driver. Debug viewer is like a text editor in which we can list the print statement of the driver coding while testing.

➢ PCI-BUS

Peripheral Component Interconnect (PCI), as its name implies is a standard that describes how to connect the peripheral components of a system together in a structured and controlled way. The standard describes the way that the system components are electrically connected and the way that they should behave.

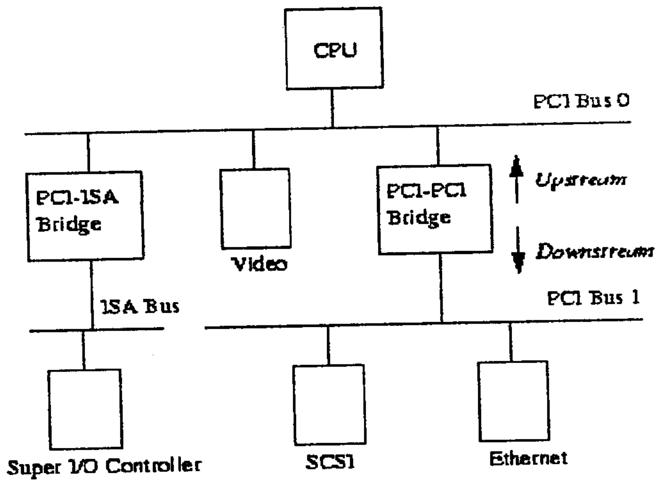
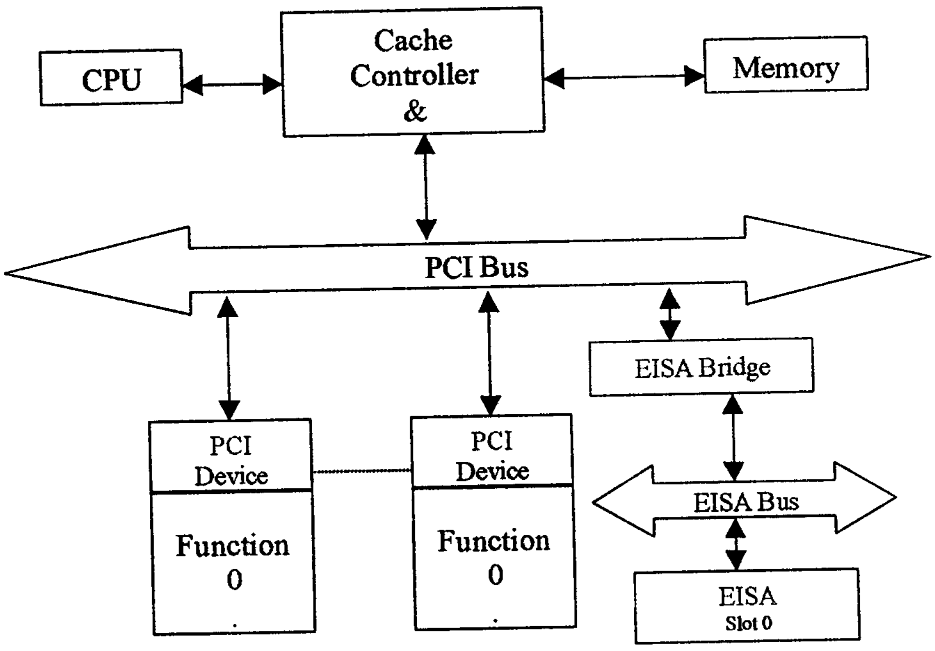


Figure: Example PCI Based System

Figure is a logical diagram of an example PCI based system. The PCI buses and PCI-PCI bridges are the glue connecting the system components together; the CPU is connected to PCI bus 0, the primary PCI bus as is the video device. A special PCI device, a PCI-PCI bridge connects the primary bus to the secondary PCI bus, PCI bus 1. In the jargon of the PCI specification, PCI bus 1 is described as being *downstream* of the PCI-PCI bridge and PCI bus 0 is *up-stream* of the bridge. Connected to the secondary PCI bus are the SCSI and Ethernet devices for the system. Physically the bridge, secondary PCI bus and two devices would all be contained on the same combination PCI card. The PCI-ISA bridge in the system supports older, legacy ISA devices and the diagram shows a super I/O controller chip, which controls the keyboard, mouse and floppy.

♦ PCI BUS DETAILS

The PCI bus is a 32 or 64-bit wide bus with multiplexed address and data lines. The bus requires about 47 lines for a complete (32-bit) implementation. The standard operating speed is 133MHz, and data can be transferred continuously at this rate for large bursts.



♦ PCI Address Spaces

The CPU and the PCI devices need to access the memory that is shared between them. This memory is used by device drivers to control the PCI devices and to pass information between them. Typically the shared memory contains control and status registers for the device. These registers are used to control the device and to read its status. For example, the PCI SCSI device driver would read its status register to find out

If the SCSI device was ready to write a block of information to the SCSI disk. Or it might write to the control register to start the device running after it has been turned on.

The CPU's system memory could be used for this shared memory but if it were, then every time a PCI device accessed memory, the CPU would have to stall, waiting for the PCI device to finish. Access to memory is generally limited to one system component at a time. This would slow the system down. It is also not a good idea to allow the system's peripheral devices to access main memory in an uncontrolled way. This would be very dangerous; a rogue device could make the system very unstable.

Peripheral devices have their own memory spaces. The CPU can access these spaces but access by the devices into the system's memory is very strictly controlled using DMA (Direct Memory Access) channels. ISA devices have access to two address spaces, ISA I/O (Input/Output) and ISA memory. PCI has three; PCI I/O, PCI Memory and PCI Configuration space.

The Alpha AXP processor does not have natural access to addresses spaces other than the system address space. It uses supporting chipsets to access other address spaces such as PCI Configuration space. It uses a sparse address mapping scheme which steals part of the large virtual address space and maps it to the PCI address spaces.

♦ PCI CONFIGURATION HEADER

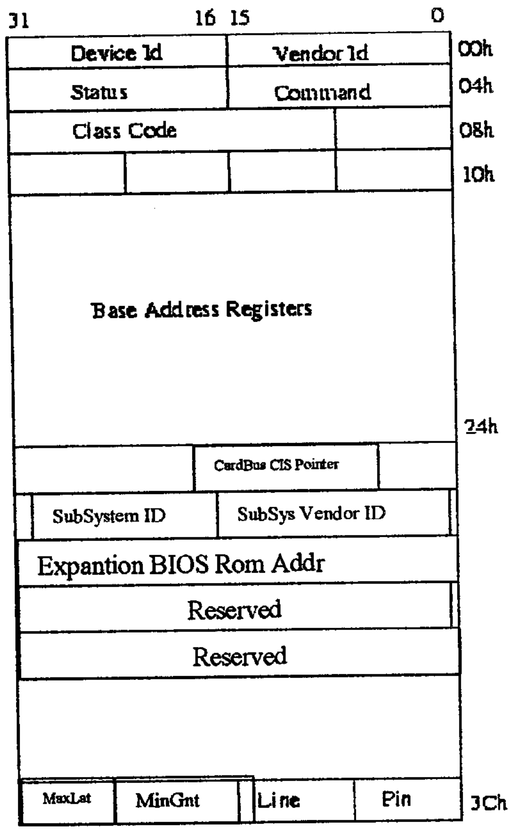


Figure: The PCI Configuration Header

Every PCI device in the system, including the PCI-PCI bridges has a configuration data structure that is somewhere in the PCI configuration address space. The PCI Configuration header allows the system to identify and control the device. Exactly where the header is in the PCI Configuration address space depends on where in the PCI topology that device is. For example, a PCI video card plugged into one PCI slot on the

PC motherboard will have its configuration header at one location and if it is plugged into another PCI slot then its header will appear in another location in PCI Configuration memory. This does not matter, for wherever the PCI devices and bridges are the system will find and configure them using the status and configuration registers in their configuration headers.

Typically, systems are designed so that every PCI slot has it's PCI Configuration Header in an offset that is related to its slot on the board. So, for example, the first slot on the board might have its PCI Configuration at offset 0 and the second slot at offset 256 (all headers are the same length, 256 bytes) and so on. A system specific hardware mechanism is defined so that the PCI configuration code can attempt to examine all possible PCI Configuration Headers for a given PCI bus and know which devices are present and which devices are absent simply by trying to read one of the fields in the header (usually the *Vendor Identification* field) and getting some sort of error. The describes one possible error message as returning *0xFFFFFFFF* when attempting to read the *Vendor Identification* and *Device Identification* fields for an empty PCI slot.

Vendor Identification	A unique number describing the originator of the PCI device. Digital's PCI Vendor Identification is *0x1011* and Intel's is *0x8086*.
Device Identification	A unique number describing the device itself. For example, Digital's 21141 fast ethernet device has a device identification of *0x0009*.
Status	This field gives the status of the device with the meaning of the bits of this field set by the standard..
Command	By writing to this field the system controls the device, for example allowing the device to access PCI I/O memory,
Class Code	This identifies the type of device that this is. There are standard classes for every sort of device; video, SCSI and so on. The class code for SCSI is *0x0100*.
Base Address Registers	These registers are used to determine and allocate the type, amount and location of PCI I/O and PCI memory space that the device can use.
Interrupt Pin	Four of the physical pins on the PCI card carry interrupts from the card to the PCI bus. The standard labels these as A, B, C and D. The *Interrupt Pin* field describes which of these pins this PCI device uses. Generally it is hardwired for a pariticular device. That is, every time the system boots, the device uses the same interrupt pin. This information allows the interrupt handling subsystem to manage interrupts from this device,
Interrupt Line	The *Interrupt Line* field of the device's PCI Configuration header is used to pass an interrupt handle between the PCI initialisation code, the device's driver and the operating system's interrupt handling subsystem. The number written there is meaningless to the the device driver but it allows the interrupt handler to correctly route an interrupt from the PCI device to the correct device driver's interrupt handling code within the operating system.

4.0 SYSTEM DESIGN

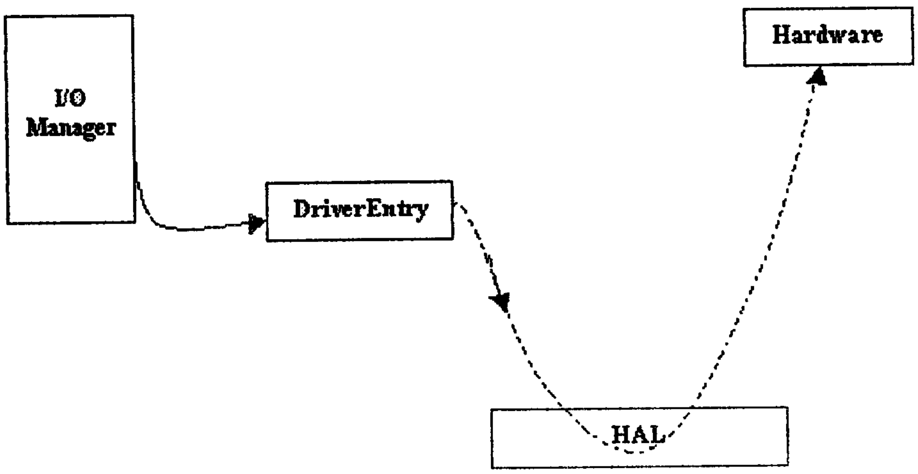
4.1 PROCESS DESIGN

The following are major functions during the device Driver Development

1. Initialization.
2. Initialization individual operations on the device.
3. Handling interrupts from the device.
4. Processing the interrupt.
5. Unloading.

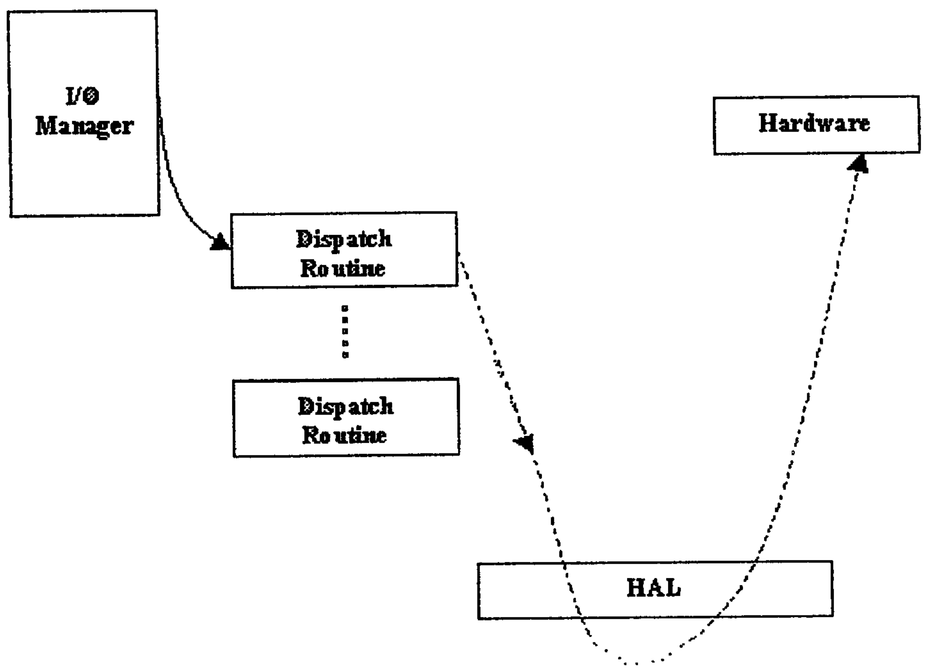
♦ **Module One:**

Initialization, the first state, is a relatively simple task. The special function called Driver Entry is called by I/O manager after the driver is loaded. Driver entry creates Device object to represent the actual hardware. A device object represents all states of the device. Driver Entry then sets up the initial state for each device and makes the device available by name to the application level.



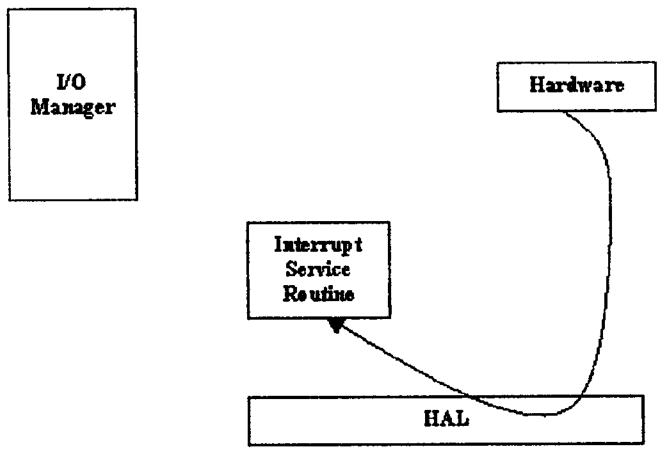
♦ **Module Two:**

The second state is to initialize the individual operations on the device. As the application code requests such operations as Open, Close, Read, and Write, NT routes these requests through the I/O manager, which calls the appropriate Dispatch routine to handle the function. Depending upon the operation, the dispatch routine may or may not touch the actual hardware. This is shown on figure below.



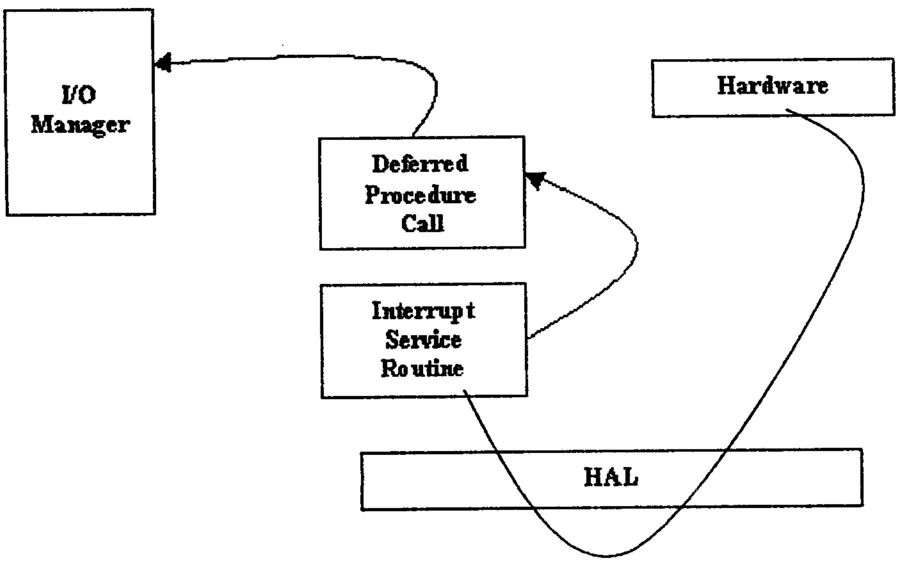
♦ **Module Three:**

The third and forth states together are one of the interesting feature of Device Driver. Handling the interrupt is done when the device notifies the processor that it needs attention. The HAL intercepts this signal and activates the Interrupt Service Routine, an entry point that is called to process the interrupt request. This routine does as little as possible- handling perhaps a dozen instructions in the simplest case – and then queues up a request for interrupt processing and returns. This is shown in figure below.



♦ **Module Four**

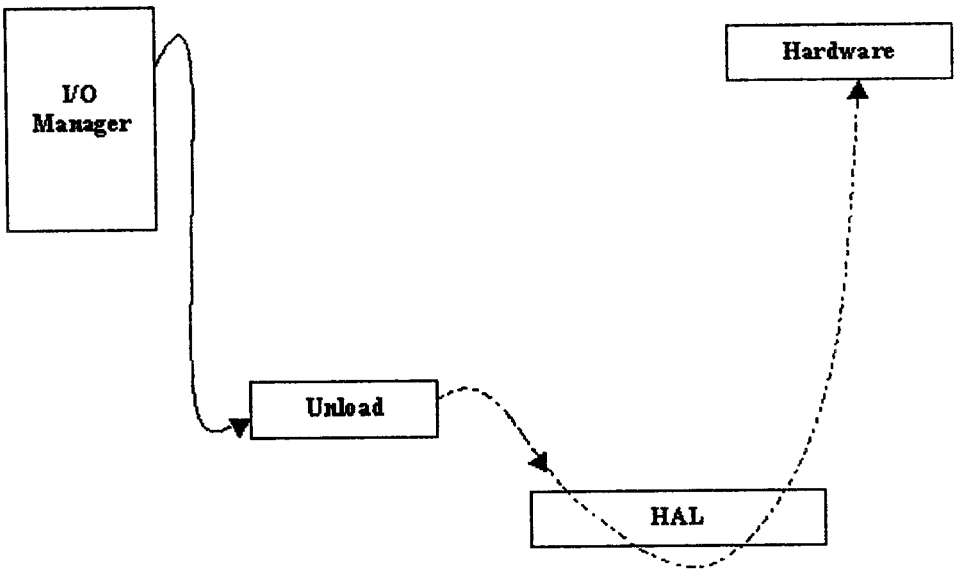
Processing the interrupt is when all of the real work of dealing with the interrupt happens. Buffers may get copied if the request was for input. If an error occurred, error recovery may be initiated if it is appropriate for the device. A new I/O operation may be started. Kernel memory that was dynamically allocated may be freed. All of these relatively lengthy operations are handled by a function that is logically "called" by the ISR but which in fact executes at some later time. Because the actual execution of the procedure is deferred, that is called Deferred Procedure Call (DPC) and the procedure itself called the DPC routine. The DPC routine is called, indirectly, by the ISR and when it completes,



it notifies the I/O manager that the I/O transfer has completed. The I/O manager can then release the application program that has been waiting for the I/O operation. The path from ISR to DPC routine to the I/O Manager is shown in figure above.

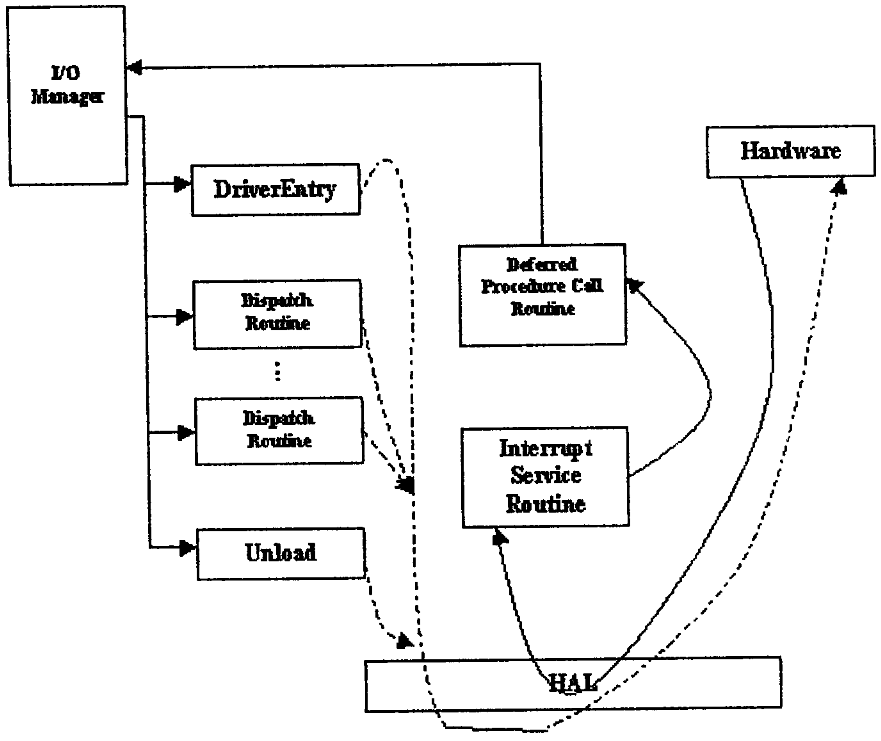
♦ **Module Five**

Ultimately, the driver needs to be unloaded. This is done in fifth state. It can be the consequence of the system's shutting down or of the drive's being explicitly stopped either by the program or by the user. The unload operation is called by the I/O manager and may or may not touch the hardware in the process of being unloaded. A device driver may, in the process of unloading, need to "shut down" the device. The structure of the Unload operation is shown in figure below.



♦ **Driver for WinNT**

An NT Device driver can be simplified to the schema shown in figure .The arrow indicates flow of control



Device Driver Entry Routine

This routine is invoked automatically by the I/O Manager when the device driver is loaded in the system . The entry routine creates and initializes system objects that are used by the I/O manager to recognize and access the driver. This is the only routine that has to be named with a fixed name: *DriverEntry*. This routine is entry point of Driver. Syntex is

```
NTSTATUS
DriverEntry (
    IN PDRIVER_OBJECT DriverObject,
    IN PUNICODE_STRING RegistryPath
    );
```

Dispatch Routines

Dispatch routines provide the main functions of the device driver. When the I/O Manager is called to perform an I/O operation, it creates an IRP and passes it to one of the driver's dispatch routines. Each dispatch routine is set up to handle a given major function code in the form IRP_MJ_XXX.

Start I/O Routine

This routine is specific to the lowest-level Windows NT device drivers. Because Windows NT is a multithreaded operating system and the I/O Manager supports asynchronous I/O operations, the I/O requests to the driver may arrive faster that it can process them. In this case queuing is required. After an I/O request has been issued, the start I/O routine is invoked first and queues the IRP representing the request for further processing by the driver dispatch routines.

Interrupt Service Routine

When a peripheral device generates an interrupt, the kernel's interrupt dispatcher transfers control to the Interrupt Service Routine (ISR) of the driver. In Windows NT, interrupt service routines run at high interrupt request levels (IRQL).

Cancel Routine

Since the Windows NT I/O system is asynchronous, a device driver may have a number of IRPs pending for an indefinite interval of time in the queue.

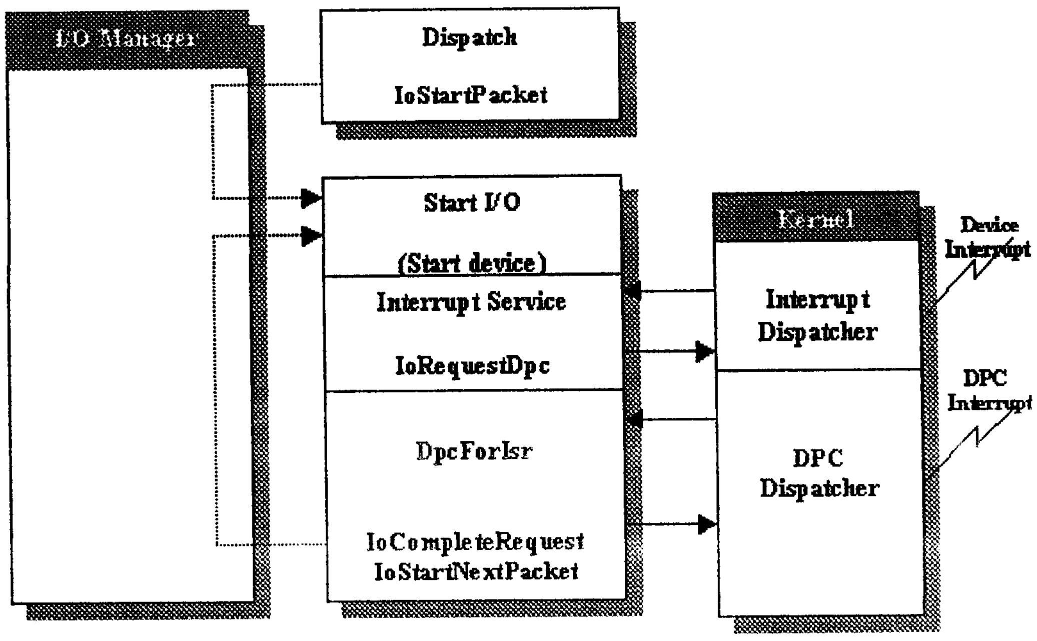
During that interval, the thread that initiated the I/O request, represented by an IRP, may be canceled.

Unload Routine

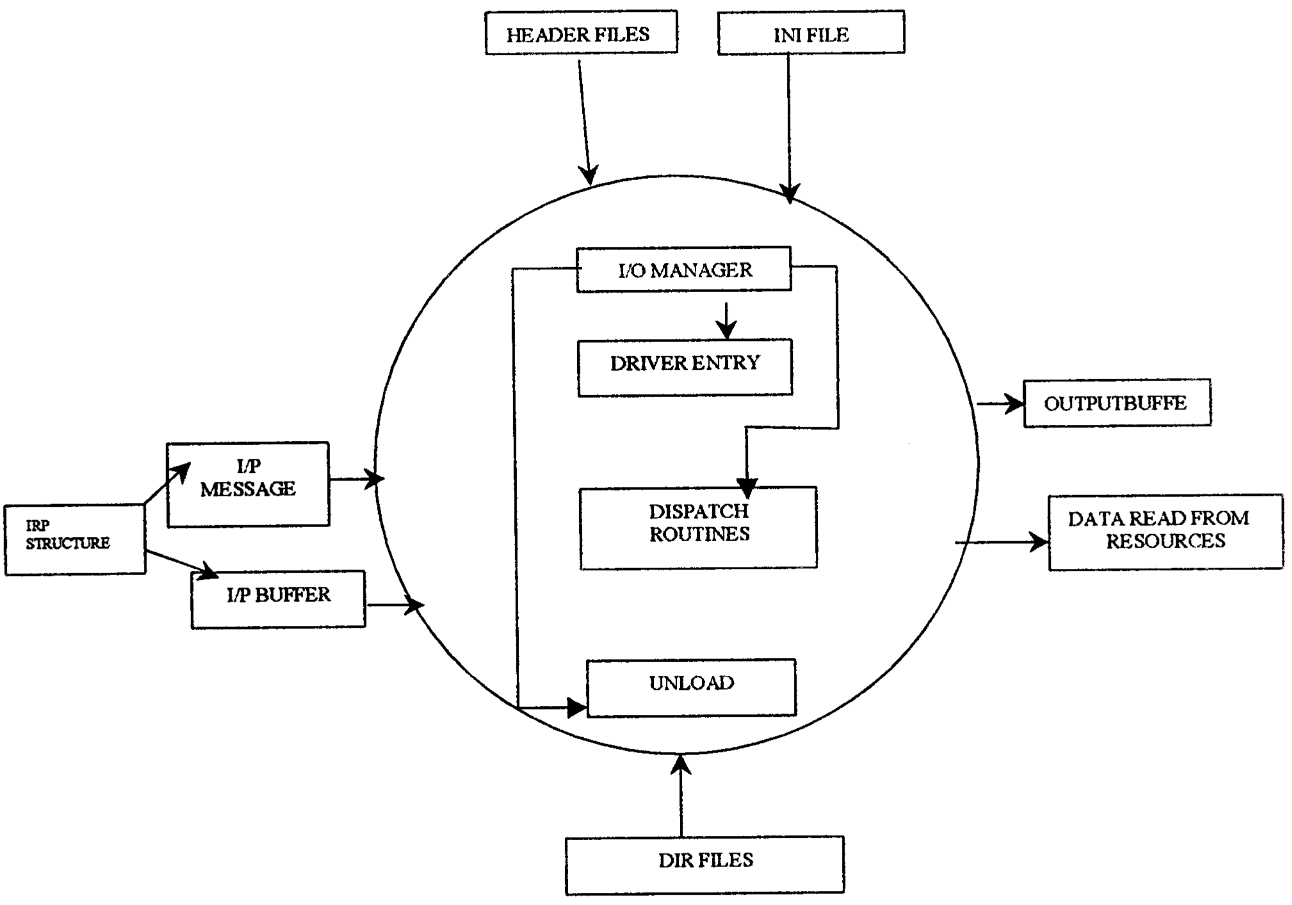
Windows NT supports drivers that can be replaced, or unloaded and reloaded while the operating system is running. Consequently, drivers must provide an unload routine. The unload routine releases all system resources associated with the device driver, so the I/O Manager can remove the driver from memory.

Forwarding an I/O Request to a Device Driver

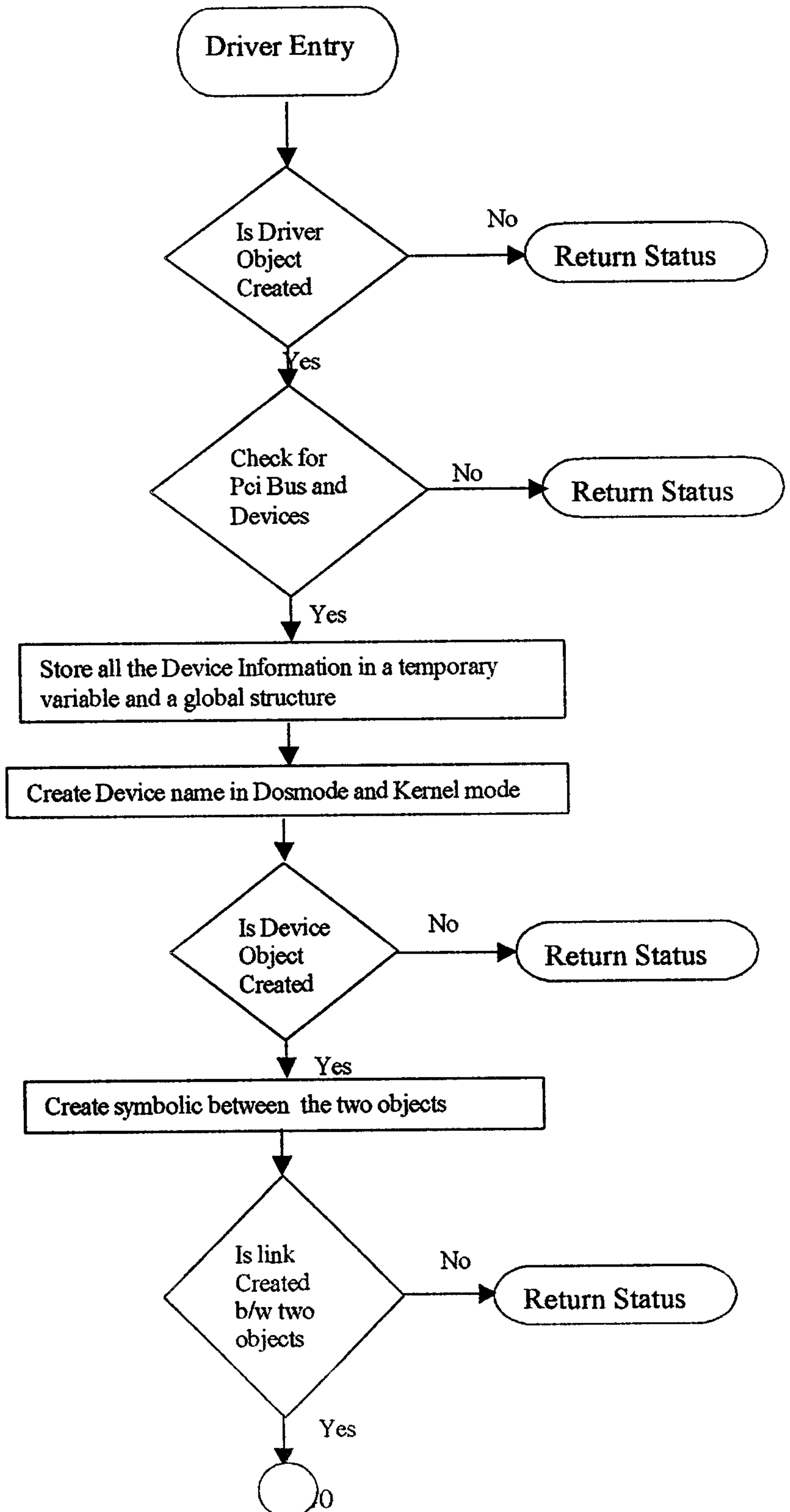
When application threads in Windows NT perform I/O operations, they obtain a file object that is a higher-level representation of the device responsible for the required I/O operations. The operating system synchronizes user threads and kernel threads by signaling the file object for I/O completion. You can get complete idea by shown figure.

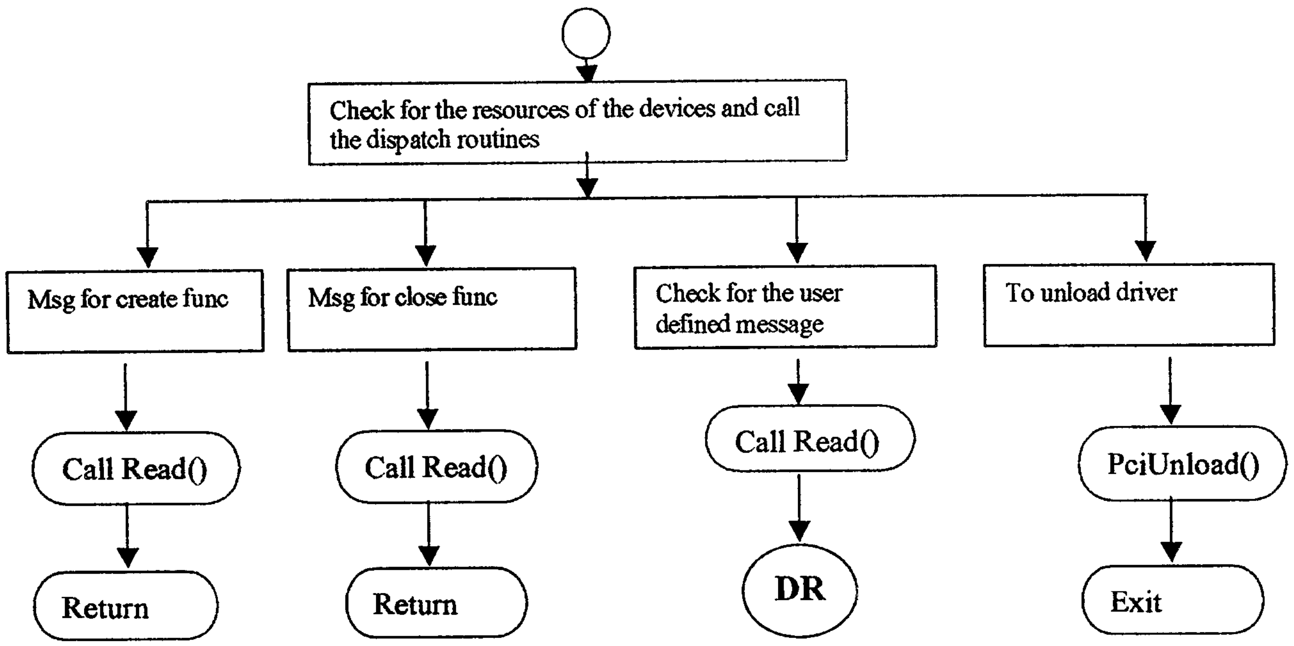


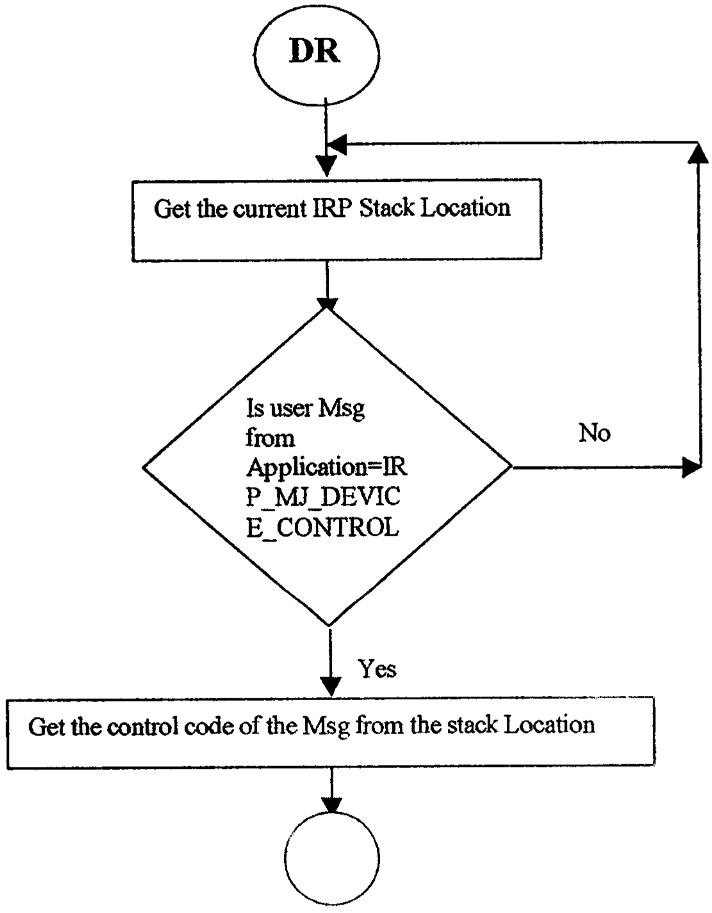
Data Flow Diagram Level 1:

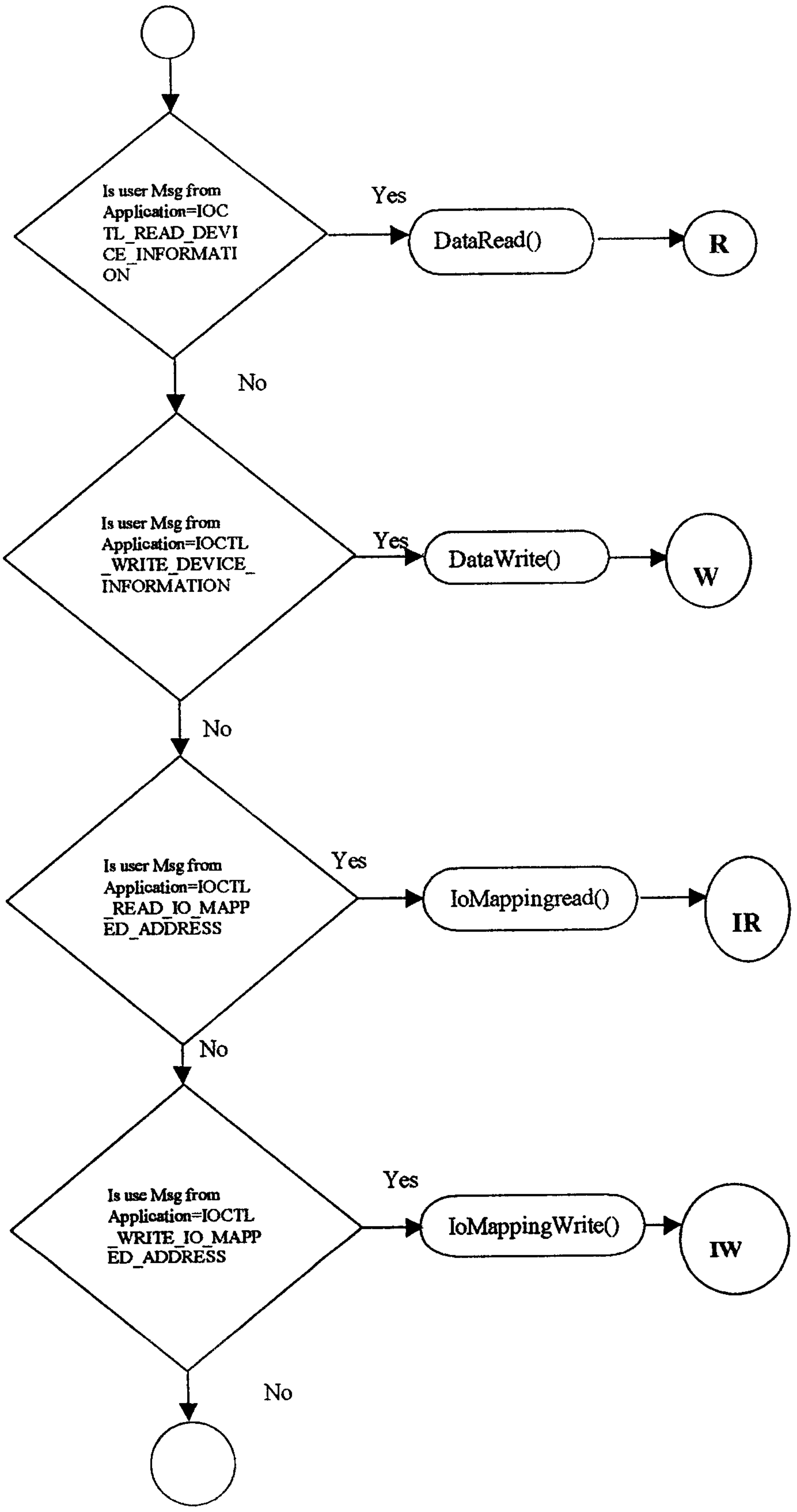


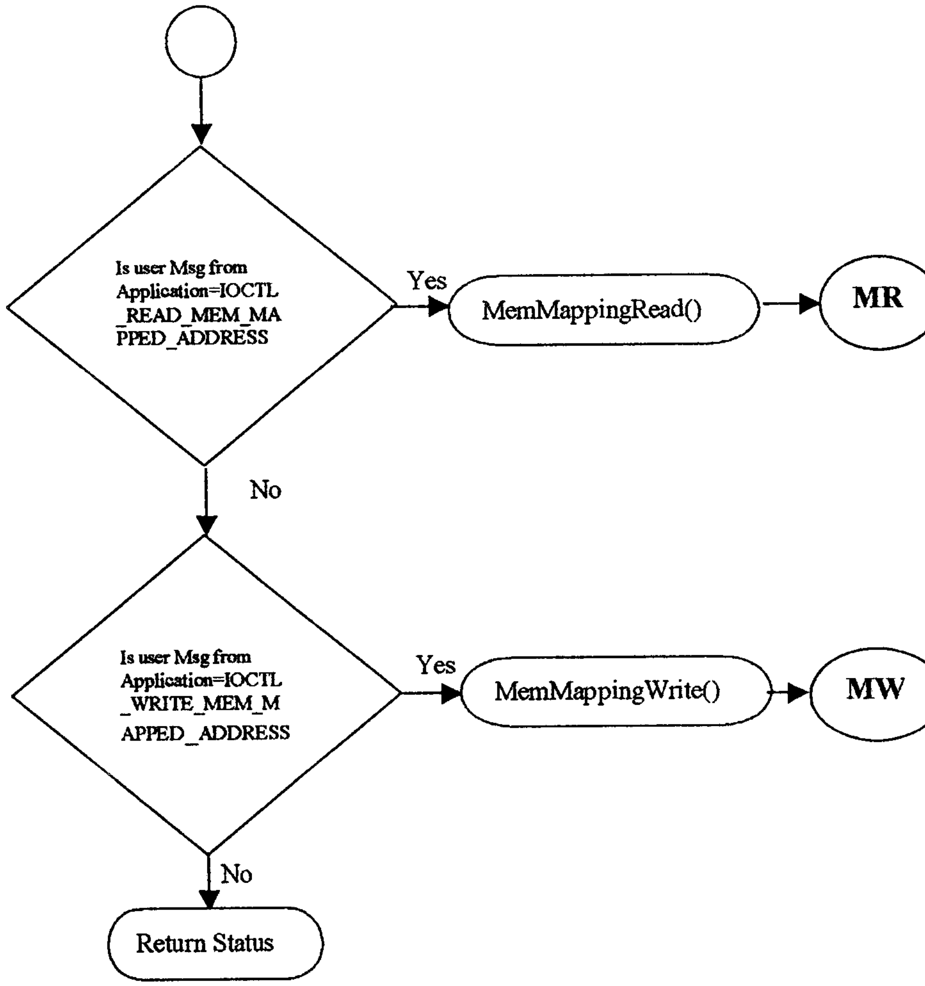
4.2 FLOW CHART

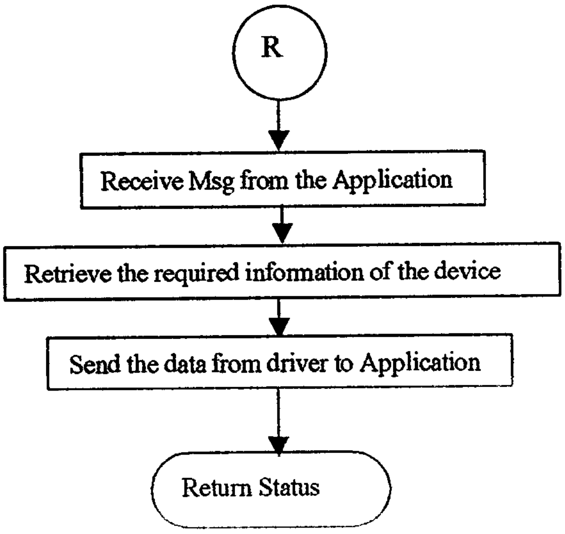

Driver Entry
Is Driver Object Created
No
Return Status
Yes
Check for Pci Bus and Devices
No
Return Status
Yes
Store all the Device Information in a temporary variable and a global structure
Create Device name in Dosmode and Kernel mode
Is Device Object Created
No
Return Status
Yes
Create symbolic between the two objects
Is link Created b/w two objects
No
Return Status
Yes
0

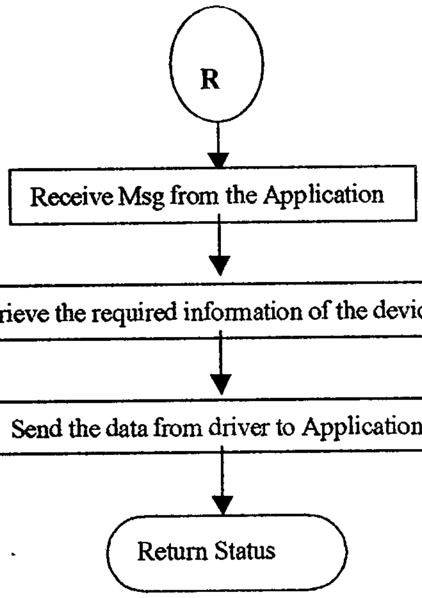

Check for the resources of the devices and call the dispatch routines
Msg for create func
Call Read()
Return
Msg for close func
Call Read()
Return
Check for the user defined message
Call Read()
DR
To unload driver
PciUnload()
Exit

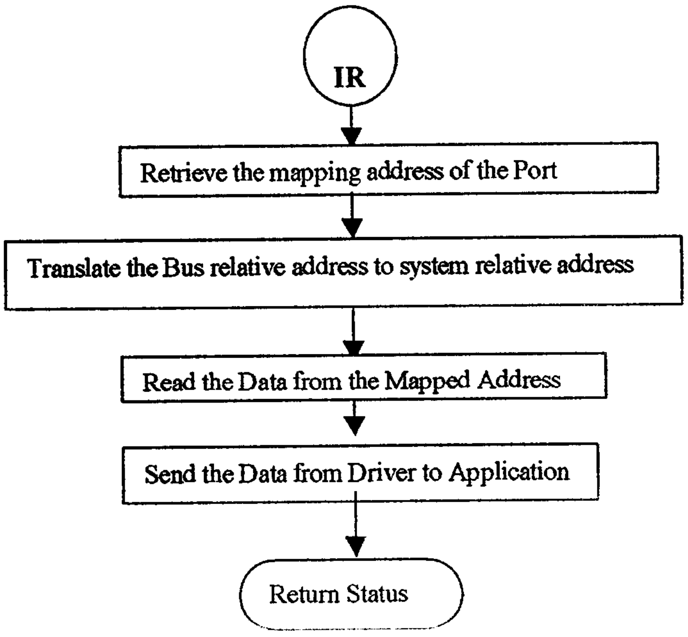

DR
Get the current IRP Stack Location
Is user Msg from Application=IRP_MJ_DEVICE_CONTROL
No
Yes
Get the control code of the Msg from the stack Location

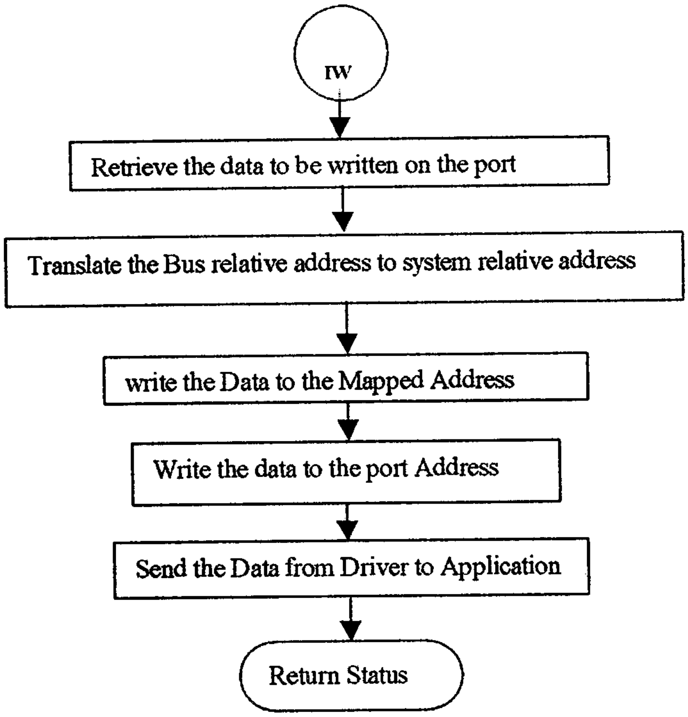

Is user Msg from Application=IOCTL_READ_DEVICE_INFORMATION
Yes
DataRead()
R
No
Is user Msg from Application=IOCTL_WRITE_DEVICE_INFORMATION
Yes
DataWrite()
W
No
Is user Msg from Application=IOCTL_READ_IO_MAPPED_ADDRESS
Yes
IoMappingread()
IR
No
Is use Msg from Application=IOCTL_WRITE_IO_MAPPED_ADDRESS
Yes
IoMappingWrite()
IW
No

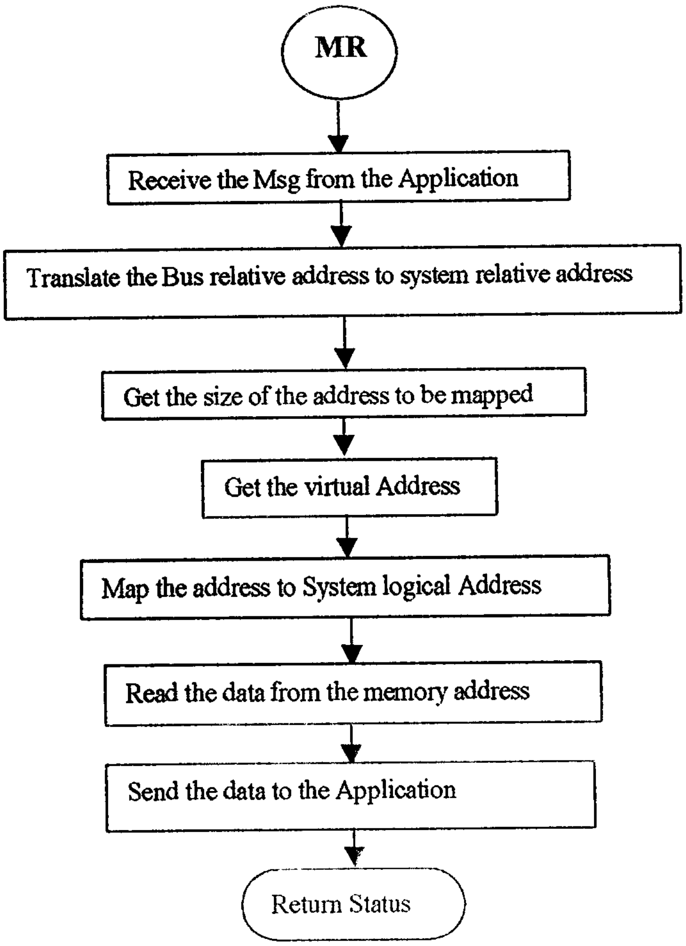

Is user Msg from Application=IOCTL _READ_MEM_MA PPED_ADDRESS
Yes
MemMappingRead()
MR
No
Is user Msg from Application=IOCTL _WRITE_MEM_M APPED_ADDRESS
Yes
MemMappingWrite()
MW
No
Return Status

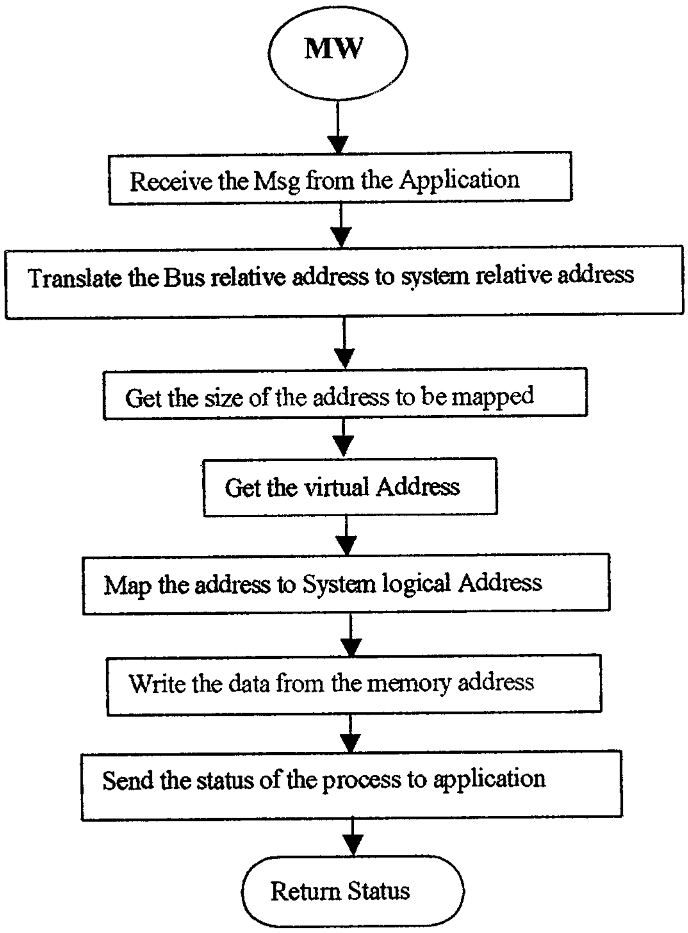

R
Receive Msg from the Application
Retrieve the required information of the device
Send the data from driver to Application
Return Status


R
Receive Msg from the Application
Retrieve the required information of the device
Send the data from driver to Application
Return Status


IR
Retrieve the mapping address of the Port
Translate the Bus relative address to system relative address
Read the Data from the Mapped Address
Send the Data from Driver to Application
Return Status


rw
Retrieve the data to be written on the port
Translate the Bus relative address to system relative address
write the Data to the Mapped Address
Write the data to the port Address
Send the Data from Driver to Application
Return Status


MR
Receive the Msg from the Application
Translate the Bus relative address to system relative address
Get the size of the address to be mapped
Get the virtual Address
Map the address to System logical Address
Read the data from the memory address
Send the data to the Application
Return Status


MW
Receive the Msg from the Application
Translate the Bus relative address to system relative address
Get the size of the address to be mapped
Get the virtual Address
Map the address to System logical Address
Write the data from the memory address
Send the status of the process to application
Return Status

5.0 SYSTEM IMPLEMENTATION AND TESTING

5.1 SYSTEM IMPLEMENTATION

Installing a device driver on a machine using the Microsoft Windows NT.

Most drivers are installed using the Windows NT 4.0 information (INF) files and Setup functions; however, some drivers (for example, network and multimedia) are installed with the legacy methods. Driver installation can occur either during or after the installation of Windows NT on the machine.

When Windows NT is installed on a machine, the initial phase of the Windows NT Setup program installs only the minimum number of drivers needed for NT to run: keyboard, mouse, video, SCSI/Disk, and Machine/HAL. This phase of the Setup program is known as text setup. If you want the user to be able to install your driver during text setup, your distribution disk must include a text file named txtsetup.oem.

After the text setup phase has been completed, the setup program boots Windows NT and proceeds with the GUI-mode phase of the Windows NT installation. During this phase, the Setup program gives the user an opportunity to install network components.

After Windows NT has been installed, a user with administrator privileges can install drivers. This would be necessary, for example, if a device is added to the hardware configuration that was not present when NT was installed or if the user wants to install a different or updated driver for an existing device. For this situation, the following installation methods are available:

- To install a display driver, keyboard driver, modem driver, mouse driver, PC Card (PCMCIA) driver, port driver, printer driver, SCSI adapter driver, tape device driver, or telephony driver, the user would run the corresponding applet in the Control Panel folder.
- To install network components, the user would run the Networks applet (ncpa.cpl) in the Control Panel folder. This applet calls the Windows NT Setup program, and expects to find an oemsetup.inf (or oemsetnt.inf) file on the supplied distribution disk.
- To install a multimedia device, the user would run the Multimedia applet (multimed.cpl) in the Control Panel folder. The driver distribution disk must contain an oemsetup.inf (or oemsetnt.inf) file
- For driver types not discussed above, the driver developer can write an installation program and provide this with the driver. The installation program would use functions in the Setup API to set up the registry and to copy files.

➢ Driver Installation Requirements:

Windows NT device driver installation includes the following components:

Copying the files required by the driver into the appropriate system directories. Storing information about the driver in the Windows NT configuration registry.

➢Text-Mode Setup:

During the text setup phase of Windows NT installation, the Windows NT Setup program installs drivers for the following components: keyboard, mouse, video, SCSI, disk, CD-ROM, and machine/HAL. If text setup cannot find a driver for any of these components, it prompts the user to insert a disk containing the driver. The user can also select other from a list of drivers for a component; this causes Setup to prompt the user for a disk.

If you are distributing a driver for one of these hardware components and intend to enable the user to install it during text setup, your distribution disk must include a file named *txtsetup.oem*. A *txtsetup.oem* file is a text file containing the following information:

- ♦ Identifies the hardware components supported by this *txtsetup.oem* file.
- ♦ Lists the files to copy from the distribution disk for this component.
- ♦ Lists the registry keys and values to create for this component.

➢INF Files:

This section describes Windows NT 4.0- and Windows 95-style INF files. *An INF file is a formatted reference file that contains information about installation files and devices such as filenames, version information, and so on.* You can create or modify an INF file using any text editor. If you are providing a Windows NT device driver to be installed with a Control Panel application or other installation program, you must create an INF file. The INF file enables your device to be installed and to work in the Windows NT environment.

➢ INF File Format Reference:

The format of the Windows NT 4.0- and Windows 95-style INF files differs from the Windows NT 3.x INF files. These new INF files do not contain installation scripts. All of the installation procedures are included in the setup program; the INF file acts as a resource, containing formatting and file information.

An INF file is made up of a set of named sections. To be used by the operating system installer, a section must contain one or more items. There can be any number of sections in an INF file.

There are approximately 20 types of sections that can be used in an INF file. Each type of section has a particular purpose; for example, to install a service or add entries to the registry.

INF files must follow these general rules:

- Sections begin with the section name enclosed in brackets.
- Values can be expressed as replaceable strings using the form %strkey%. To use a % character in the string, use %%. The strkey must be defined in a **Strings** section of the INF file.

Each INF file must contain a **Version** section identifying it as a Windows NT 4.0/Windows 95-compatible file.

5.2 SYSTEM TESTING

Windows NT DDK drivers are built using the build utility, which uses a set of rules and project files that specify how drivers should be created. The build -cef command normally used to build drivers causes a scan of dependency files, performs a clean build, and creates an error log.

The dirs file specifies which directories in the sub trees contain source code files to be built. The sources file specifies which source files are required to build the current driver.

If your driver will consist of multiple binaries, or the source will be kept in multiple directories, you might need to create a dirs file specifying which directories are to be built and in what order. Each separate binary requires a sources file describing which files are to be compiled/linked to create the driver.

Each driver directory also contains a makefile. Build spawns the nmake utility for each source file listed in sources and nmake uses the makefile to generate dependency and command lists. The standard makefile file in a driver source code directory directs nmake to the master nmake macro definition file, makefile.def. This file defines the flags for the build tools such as the compiler and linker. Makefile.def simplifies the creation of platform-independent driver projects and is similar to the ntwin32.mak file in the Win32 SDK.

After you have dirs, sources, and makefile files created, the next step is to run the build utility which parses the sources file and spawns nmake for each source file. The nmake utility evaluates the macros in makefile.def and spawns the C compiler with the proper switches. After the compile stages are completed, build spawns nmake again to link the objects and complete the driver building phase.

If your sources file is correct, running build -cef is all that is required to compile and link your driver. Using build and makefile.def removes the guesswork, of which compiler switches are required, which arguments the linker requires, and so on. By adding the appropriate defines to the sources file, it is possible to control the build options in a platform-independent fashion.

During driver development you will need to build free and checked versions of your driver. This is controlled by environment variable settings that are interpreted by build and nmake. These variables are set by setenv.bat, as discussed previously. To build a free driver, run build from the free environment. To build a checked driver, run build from the checked environment.

Template files (.tpl) for dirs, sources, makefile, and other files can be found in the \<destination>\doc directory. Working versions of these files can be found in driver source subdirectories of the DDK. Examining these files along with makefile.def (located in the \<destination>\inc directory) provides additional information.

6.0 CONCLUSION

The approach of this project provides us a methodology to write a single driver for multiple devices, which can be very feasible for the multiple application integration. The driver developed will provide the full-required functionality of a memory, I/O and interrupt operation on any PCI device. The driver provides an interrupt service routine for handling hardware interrupts. The driver also provides routines for handling read () and write () system calls

The goals achieved by developing this device driver are

- Performing Memory management
- Capable of Handling interrupts
- Reducing the access time of devices
- Getting access to multiple devices using a single driver
- Further enhancement is also possible

7.0 SCOPE FOR FURTHER DEVELOPMENT

- The device driver developed for PCI-BUS can be expanded.
- Any user device driver like driver for CD-ROM or MOUSE can be added with this driver if they are connected to the PCI-BUS.

8.0 BIBLIOGRAPHY

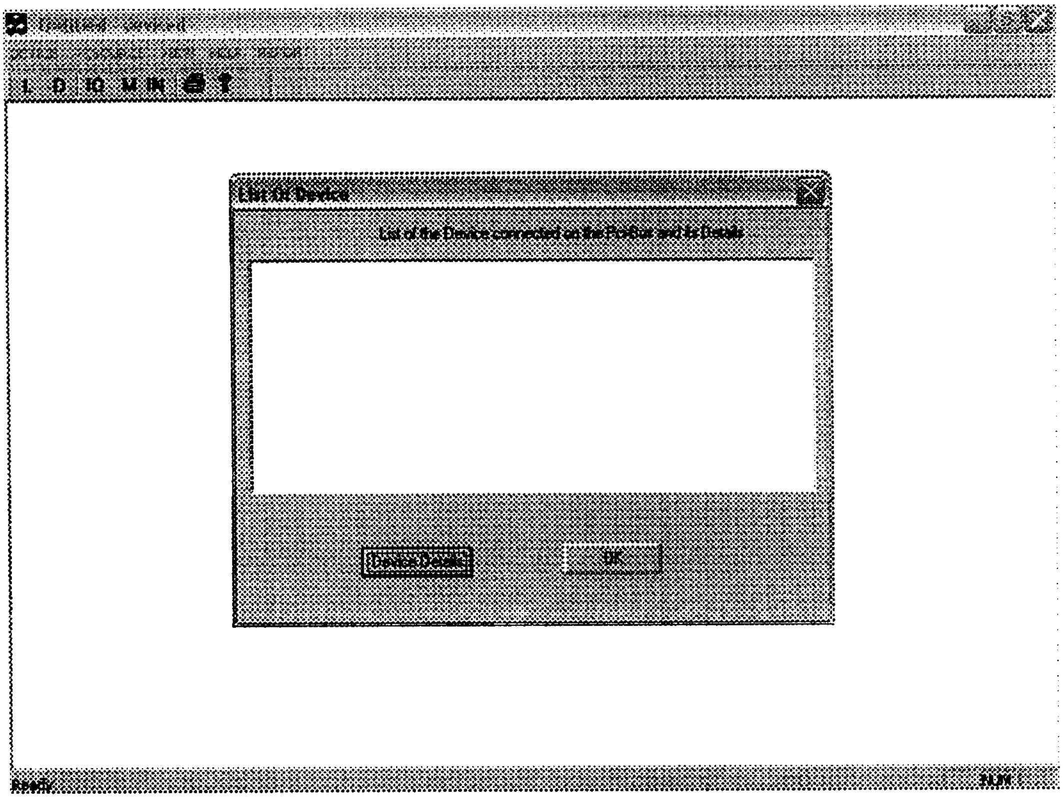
Books Referred:

- William Stallings, 'Operating Systems', Prentice-Hall India, Eastern Economy Edition (second edition), 2001.
- Peter G. Viscarola & W. Anthony Mason. "Windows NT Device Driver Development ", Macmillan Technical Publishing, 1998.
- Edward N. Decker & Joseph M. Newcomer, "Developing Windows NT Device Driver", Addison Wesiey Longman,Inc , Microsoft Edition 1998, March 1999.
- E.Balagurusamy, 'Object-Oriented Programming', Tata McGraw Hill Publication, 1999, Page No 1-25, 97-114.
- 'MSDN (Microsoft Developers Network)', Microsoft Corporation, Visual Studio 6.0 Release.
- David J. Kruglinski & George Shepherd & Scot Wingo, "Programming Microsoft Visual C++ Fifth Edition- Microsoft Press ", WP Publishers & Distributors Pvt. Ltd.
- Mickey Williams, "VC++ in 21 days", SAMS Techmedia, Second Edition 2001.

9.0 APPENDIX

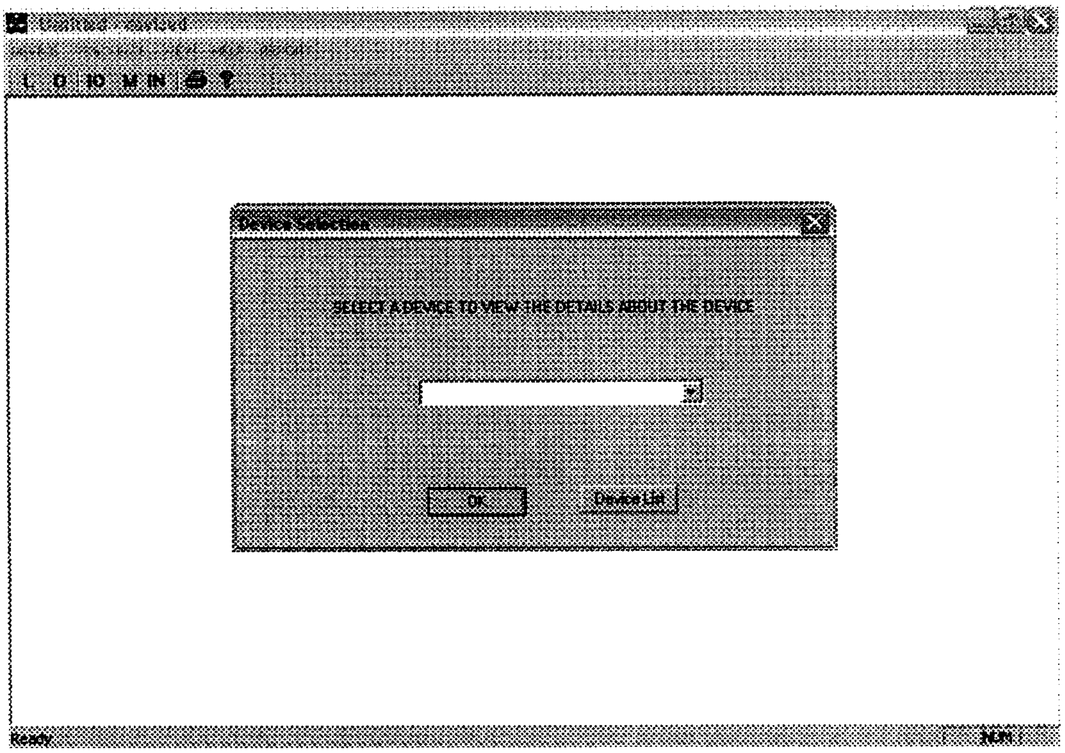
9.1 SAMPLE SCREEN

LIST OF DEVICE



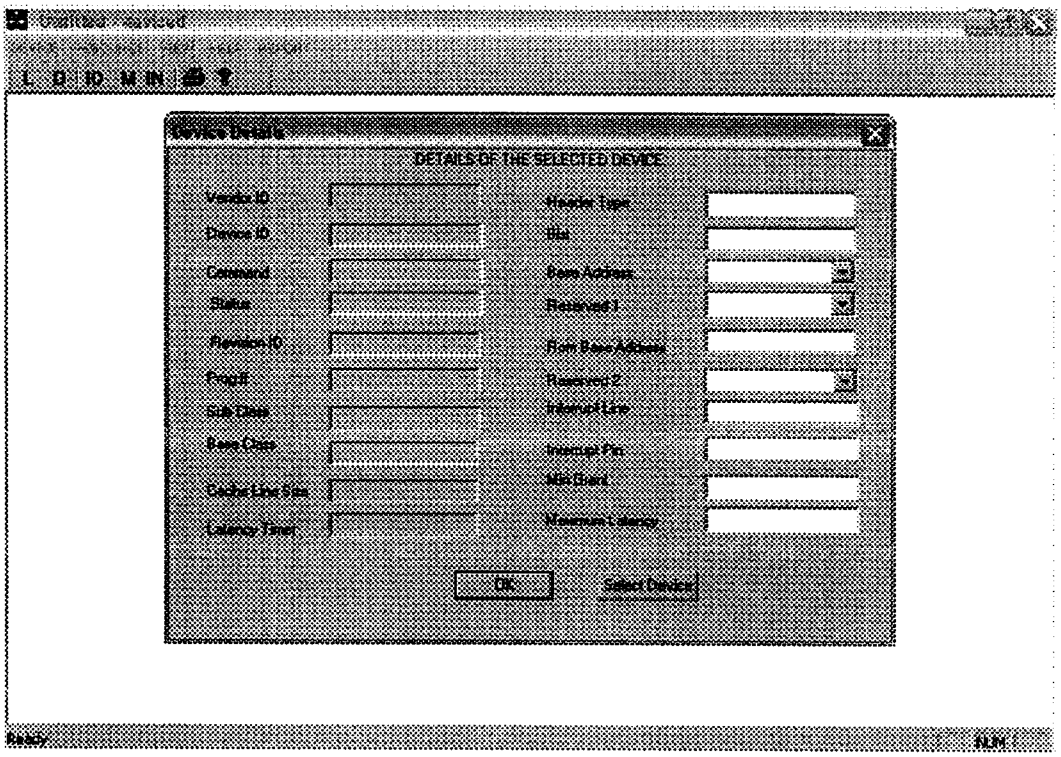
List of the Device connected on the ProBus and its Details

SELECTING DEVICE NUMBER



SELECT A DEVICE TO VIEW THE DETAILS ABOUT THE DEVICE

DEVICE DETAILS

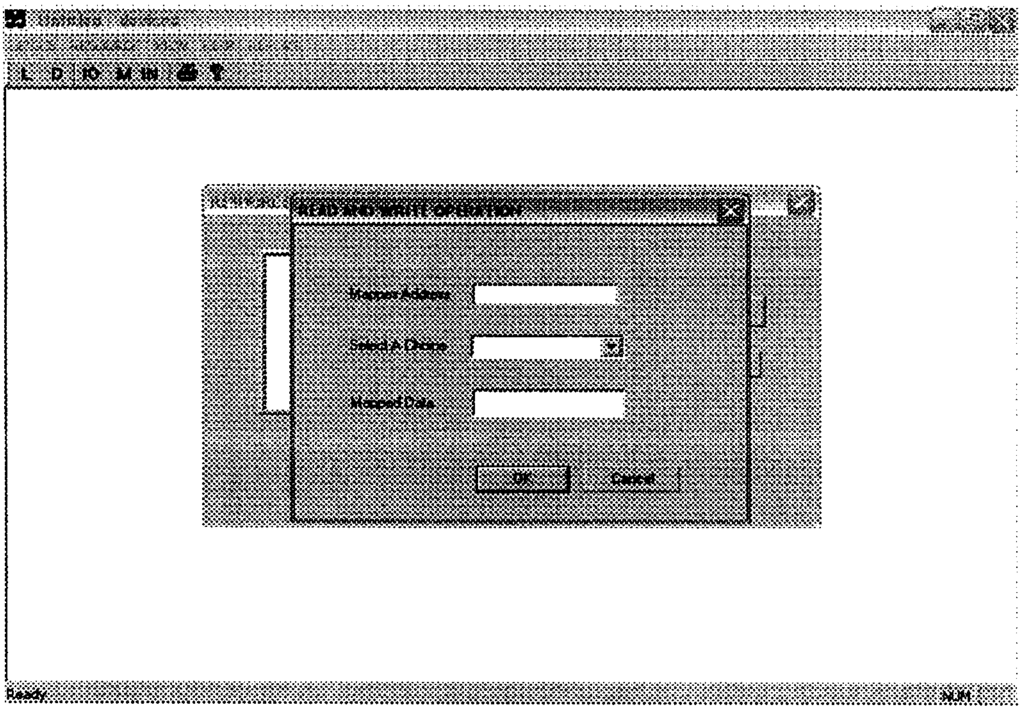


Device Details

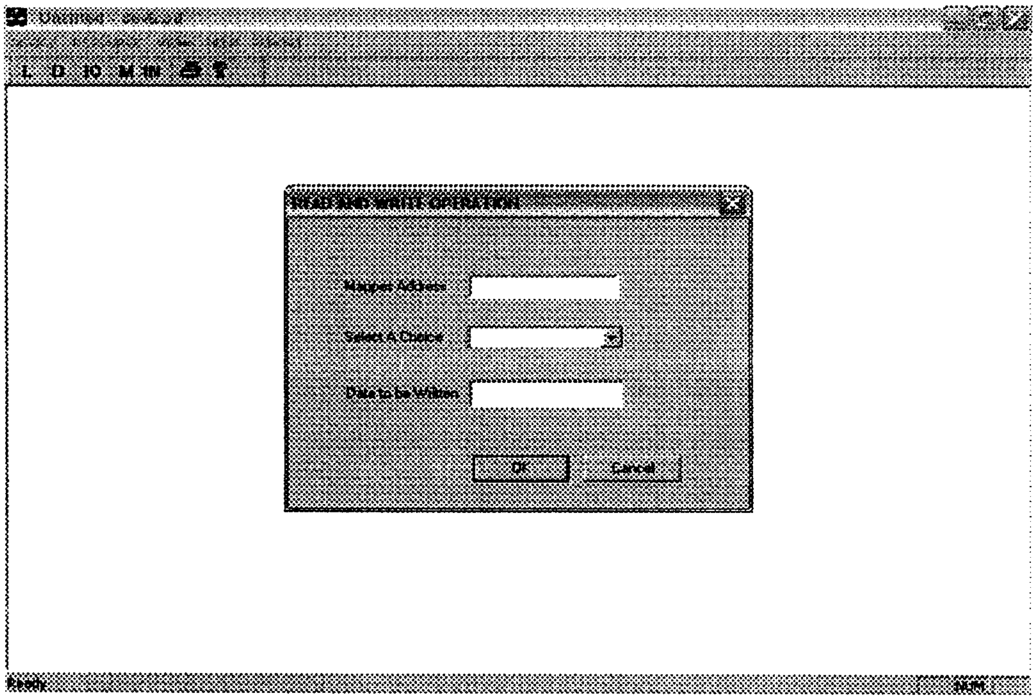
DETAILS OF THE SELECTED DEVICE

Vendor ID		Header Type	
Device ID		Bist	
Command		Base Address	
Status		Reserved 1	
Revision ID		Rom Base Address	
Prog If		Reserved 2	
Sub Class		Interrupt Line	
Base Class		Interrupt Pin	
Cache Line Size		Min Grant	
Latency Timer		Maximum Latency	

READ OPERATION



WRITE INFORMATION



9.2 SAMPLE CODE

```
CDevicedView::CDevicedView()
{

}

CDevicedView::~CDevicedView()
{

}

void CDeviceList::OnOK()
{

        CDialog::OnOK();
         CDeviceSelection DeviceSelection;
         DeviceSelection.DoModal();

}


void CInterrupt::DoDataExchange(CDataExchange* pDX)
{

         CDialog::DoDataExchange(pDX);

}


void CResourceInformation::OnWrite()
{


            CDialog::OnOK();

}
```