UNIT 3: USER INTERFACE DESIGN PROGRAMMING TOOLS

Table of contents

1.0 INTRODUCTION

The programming tools for the design of user interface give implementation support for the levels of services for programmers. These include the windowing systems that provide the core support for separate and simultaneous user-system activity. They enable easy programming of the application and the control of dialogue between the system and the user. The interaction toolkits for example, bring programming closer to the level of user perception while the user interface management systems control the relationship between the presentation and functionality.

2.0 OBJECTIVES

By the end of this unit, you should be able to:

- Explain the various levels of programming support tools
- Utilize toolkits for programming interaction objects
- Understand the concepts of the User Interface Management Systems(UIMS)

3.0 MAIN CONTENT

3.1 How Human Computer Interaction affects theprogrammer

Advances in coding have elevated programming through hardware that specifically improves upon the programmer's Interaction-technique.

The layers of development tools, as earlier mentioned, also contribute to how human computer interaction affects the programmer. These tools incorporate the windowing systems, the interaction toolkits and the user interface management systems as exemplified in the following:

Levels of programming support tools

- Windowing systems
  - device independence
  - multiple tasks
- Paradigms for programming the application
  - read-evaluation loop
  - notification-based
- Toolkits
  - programming interaction objects
- UIMS
  - conceptual architectures for separation
  - techniques for expressing dialogue

3.1.1 **Elements of the Windowing Systems**

Device independence

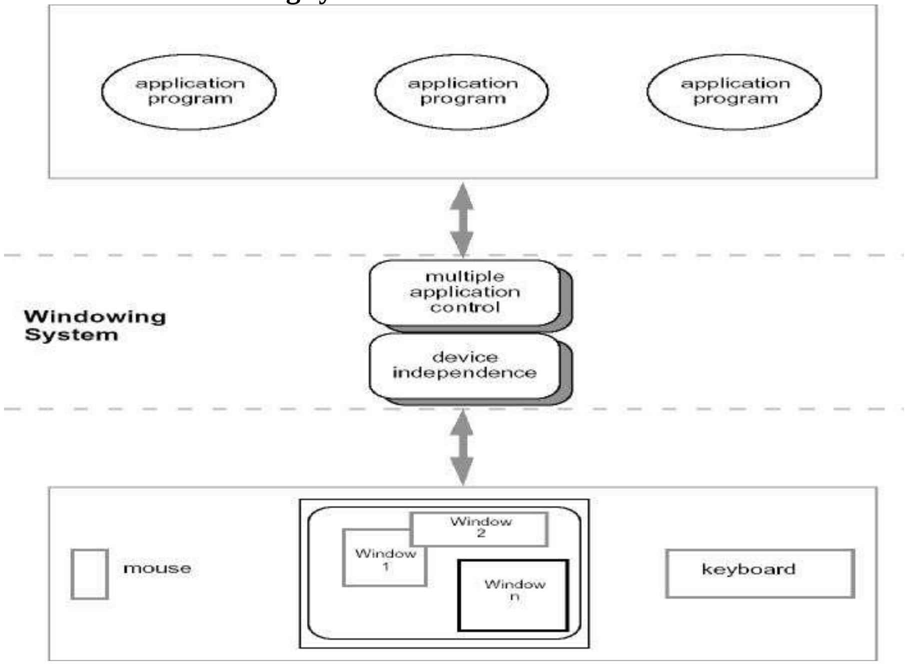
Programming the abstract terminal device drivers using the image models for output and input is device independent. Also device independence is the creation of the image models for output and the input, partially. These image models are the pixels, the PostScript (as in Macintosh Operating System X and NextStep), the Graphical Kernel System (GKS) and the Programmers' Hierarchical Interface to Graphics (PHIGS)

Resource sharing

Another element of the windowing system is resource sharing. This is the act of achieving simultaneity of user tasks. Resource sharing enables the use of the window system to support independent processes by the isolation of individual applications.

Elements of windowing systems

3.1.2 Role of a windowing system



As shown in the diagram above, the windowing system comprising the multiple application control and the device independent control enables the interface between the application programs and the user.

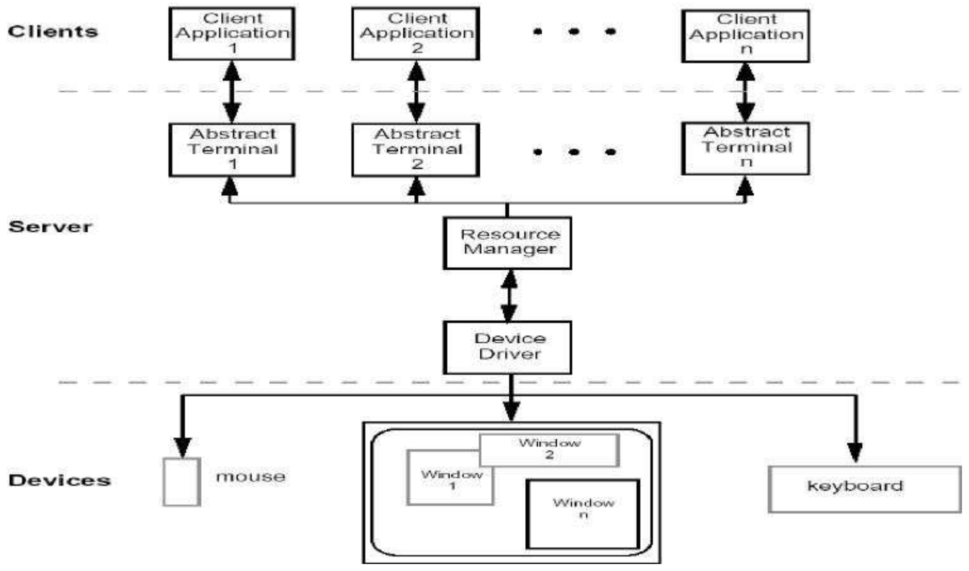
3.1. 3 The Architectures of windowingsystems

The Architectures of windowing systems are analysed through three possible software architectures if we all assume device driver is separate and know how they differ and how the multiple application management is implemented.

The three possible software architectures are in the following forms:

1. When each application manages all processes. Here, everyone worries about synchronization and reduces portability of applications
2. When management role within kernel of operating system ensures that applications are tied to operating system, and
3. When management role as separate application ensures maximum portability The

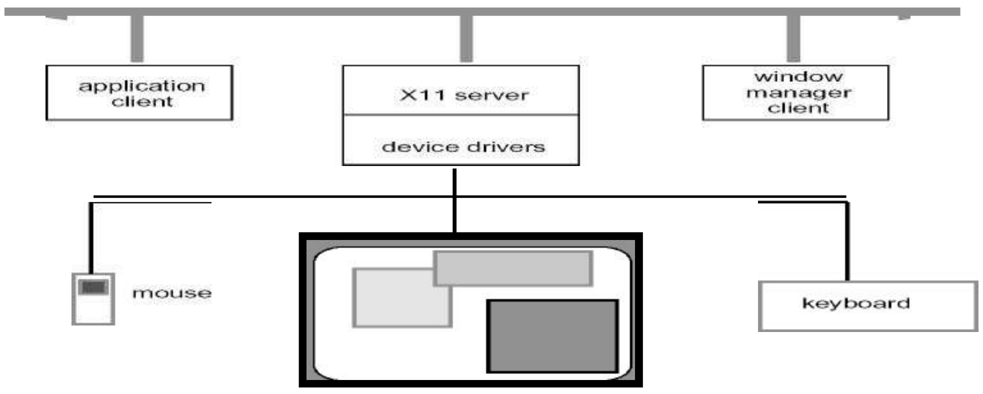
client-server architecture is illustrated below:



3.1.4 X Windows architecture

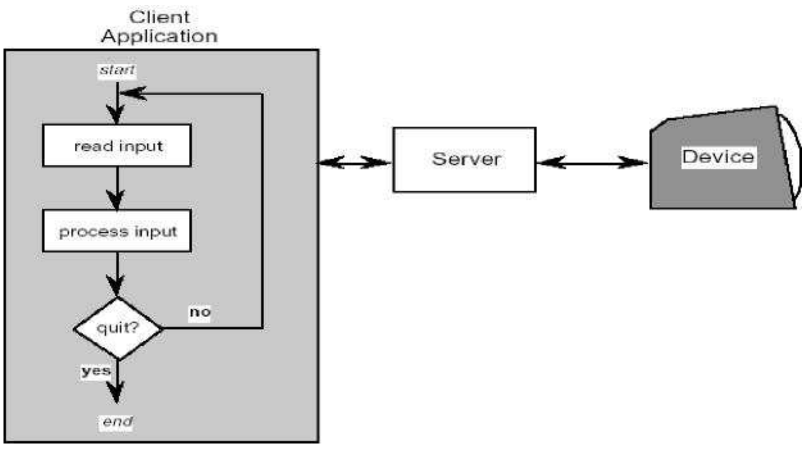
The X Windows architecture comprises the Pixel imaging model with some pointing mechanism and the X protocol that defines the server-client communication.

The architecture also contains a separate window manager client that enforces policies for input and output. Policies on how to change input focus, how the tiled windows compare with overlapping windows and policies on inter-client data transfer. See the pictorial illustration below.



3.1.5 **Typical program models of the application:**

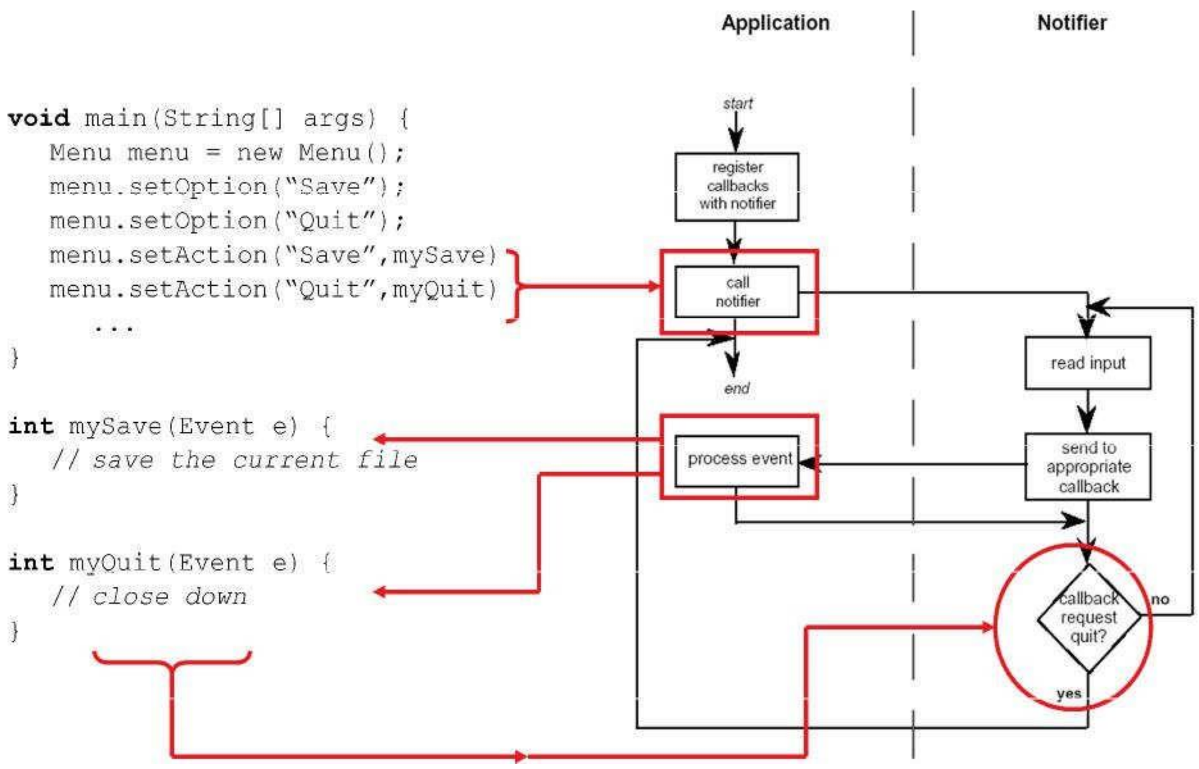
Programming the application - 1



A typical read-evaluation loop is provided below: repeat read-event(myevent) case myevent.type

```
type_1:
do type_G1 processing
type_2:
do type_G2 processing
type_n:
do type_Gn processing
        end case end repeat
```

Programming the application - 2 Notification-based



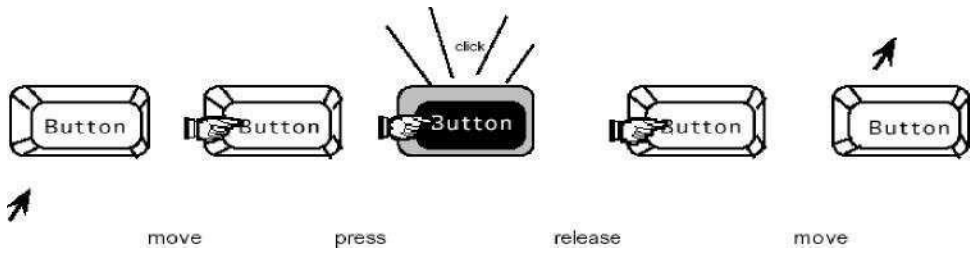
```
void main(String[] args) {
   Menu menu = new Menu();
   menu.setOption("Save");
   menu.setOption("Quit");
   menu.setAction("Save",mySave)
   menu.setAction("Quit",myQuit)
      ...
}

int mySave(Event e) {
   // save the current file
}

int myQuit(Event e) {
   // close down
}
```

.0 USINGTOOLKITS

3.2.1 User Interface Toolkits

User interface toolkits are probablyt three major platforms

he most widely used tool nowadays to impl ment applications. All (Unix/Linux, MacOS, and Windows) c ome with at least one standard UI toolkit. Toolkits are interaction objects withinput and output intrinsically linked. They enable programming with interaction techniques using widgets similar look andfeel.
A widget is a software object that ha behavior, and an application interface
and gadgets. They promote consistency and generalizations through s three facets that closely match the MVC m odel: a presentation, a e. Sample illustration of widgets is provided below.



The presentation defines the graphical aspect of the widget. The overall presentation of an interface is created by assembling widgets into atree. Widgets, such as buttons are the leav s of the tree. Composite
widgets constitute the nodes of the t ree and control the layout of their children. The behavior of a widget
defines the interaction methods it supports: a button can be pressed, a scrollbar can be scrolled, and a text field can be edited.
The application interface defines how a widget communicates the results of the user interaction to the rest of the application. It is usually based on a notification mechanism.
One limitation of widgets is that their behaviors are limited to the widget itself. Interaction techniques that involve multiple widgets, such as drag-and-drop, cannot be supported by the widgets' behaviors alone and require separate support in the UI toolkit.

3.2.2 Prototypes and Widgets

In general, prototyping new interaction techniques requires either implementing them within new widget classes, which is not always possible, or not using a toolkit at all. Implementing a new widget class is typically more complicated than implementing the new technique outside the toolkit, (for example, with a graphical library), and is rarely justified for prototyping. Many toolkits provide a blank widget, such as the Canvas in Tk or JFrame in Java Swing, which can be used by the application to implement its own presentation and behavior. This is usually a good alternative to implementing a new widget class, even for production code.

A number of toolkits have also shifted away from the widget model to address other aspects of user interaction. For example, GroupKit was designed for groupware, Jazz for zoomable interfaces, the Visualization and InfoVis toolkits for visualization, Inventor for 3-D graphics, and Metisse for window management.

Creating an application or a prototype with a UI toolkit requires solid knowledge of the toolkit and experience with programming interactive applications. In order to control the complexity of the interrelations between independent pieces of code (creation of widgets, callbacks, global variables, etc.), it is important to use well-known design patterns, otherwise, the code quickly becomes unmanageable and, in the case of a prototype, unsuitable to design models.
Toolkits are amenable to object-oriented programming using Java interfaces that include the AWT (abstract windowing toolkit), a Java toolkit, and some Java classes for buttons and menus, etc.
Some Java interfaces are (i) Notification based such as AWT 1.0 with the need to subclass basic widgets and AWT 1.1 and beyond with call-back objects (ii) Swing toolkit built on top of AWT with higher level features that also uses the MVC architecture.

3.2. 3 The User Interface Management Systems (UIMS)

The UIMS add another level above toolkits because toolkits may be too difficult for non-programmers. Concerns of UIMS include the conceptual architecture, the implementation techniques and the support infrastructure

UIMS as conceptual architecture

The conceptual architecture is viewed as the *separation between application semantics and presentation. This improves:*

i. Portability which runs on different systems

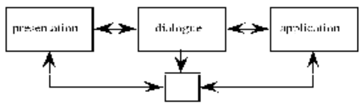
ii. Reusability having components reused thereby cutting costs

iii. Multiple interfaces that access same functionality

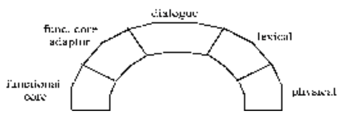
iv. Customizability; here, the system is customised to suit the designer and user.

The User Interface Management System (UIMS) tradition of interface layers and logical components comprise

- linguistic: lexical/syntactic/semantic
- Seeheim:

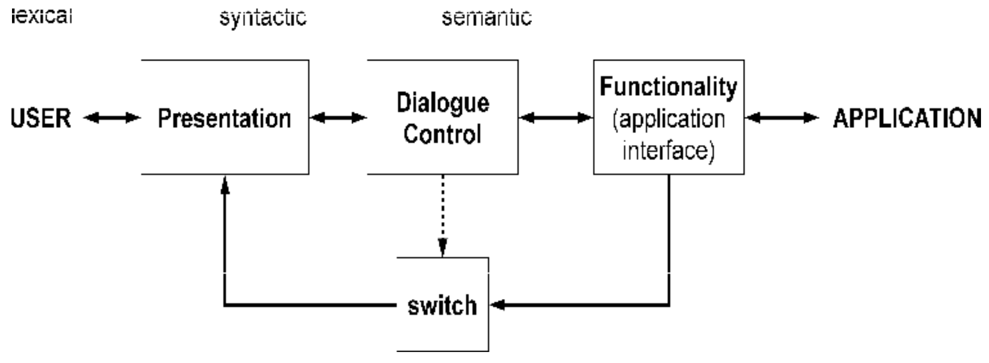


- Arch/Slinky



The Seeheim model: Its concept and implementation

Seeheim concept arose as a result of implementation experience with conceptua l approach as principal contribution. The concepts are part f the normal user interface language.



The above depicts different kinds of feedback. For example, the movement of th e mouse carried out at the
presentation interface is known as th lexical feedback, the menu highlights as a dialogue control is known as the syntactic feedback while a function carried out at the application interface changing ,is such as sum of number
regarded as the semantic feedback.

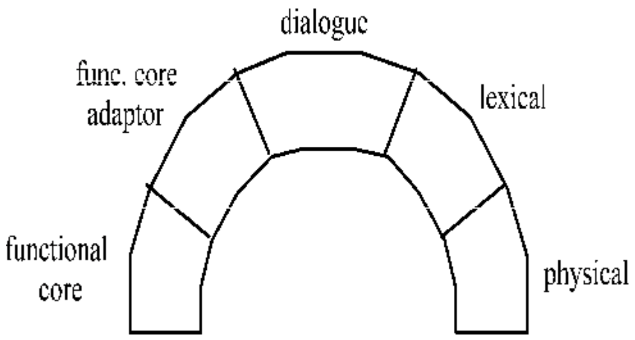
Because the semantic feedback is often slower, programmers prefer to use the rapid lexical and/or the syntactic feedbacks.

ss switch is needed for implementation. The switch enables a direct
The lower box representing the bypa and presentation.

Though regulated by a di communication between application
provides a rapid semantic feedback.
logue control, it also

The Arch/Slinky model characteristics

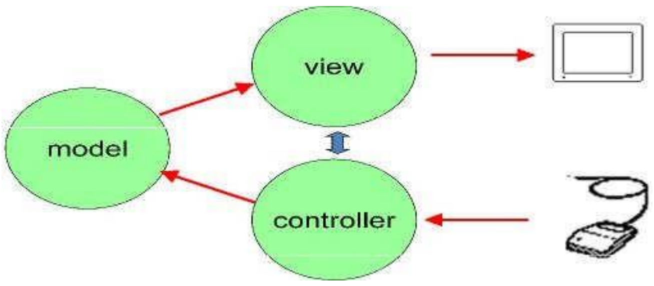
- This model contains more layers to distinguish the lexical and the physical
- Like a 'slinky' spring, different layers may be thicker (that is more important) in different systems or in
different components



Monolithic vs. Components
Seeheim has big components and is often easier to use smaller ones especially. if using object-oriented toolkits .

Smalltalk used the model—view—cont roller (MVC) Model indicates the internal logical state of component View shows how it is rendered on screen
Controller processes user input

The Model - View - Controller (MVC)



Input

control

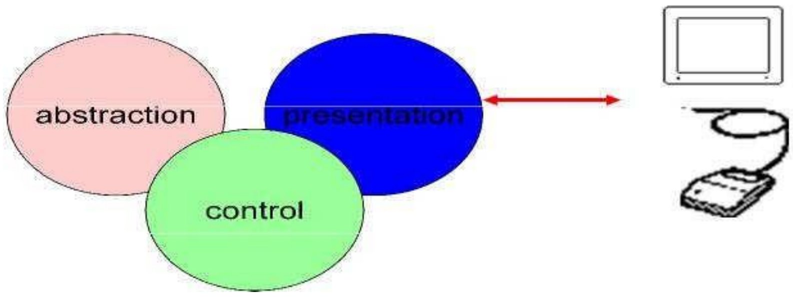
model

view MVC

issues
MVC is largely pipeline model:
Using the pictorial illustration above, the Input is transmitted to the controller, th e
controller processes user input and connects with the mo del. Since the model represents the internal the logical state of the

component, a manipulation is carried
out and through the view model, the result of the manipulation is
output on screen. In graphical interface, input only has meaning in relation to output e.g. a mouse click There is the need to know *what* was clicked and the controller has to decide what to do with the click. Using the internal logical state of the component through the model, the view di plays how it is rendered on the screen.
However, in practice, the controller directly 'talks' to view.

The Presentation, Abstraction and Control (PAC) model presentation - abstraction - control



The PAC model is closer to the Seeheim model in principle because the term 'Presentation' describes input and output are managed, the ' Abstraction' describes the logical state of th component while 'Control' represents the state of mediation between the 'Presentation' and 'Abstraction' .
The PAC model manages the hierarcy and multiple views through the control pa rt of PAC communicate the 'Abstraction'
Though the PAC model is direct, the MVC model is used more in practice as can b e found in the use of Java Swing.

The Implementation of UIMS takes the following forms:
Implementing the techniques for dialogue controller through the use of the

- Menu networks
- State transition diagrams
- Grammar notations
- Event languages
- Declarative languages
- Constraints
- Graphical specification

4.0 CONCLUSION

Programming tools for the design of user interface enable easy programming of the application and the control of dialogue between the system and the user.

5.0 SUMMARY

Levels of programming support tools comprise Windowing systems that are device independence with multiple tasks, the Paradigms for programming the application with read-evaluation loop that is notification-based , Toolkits containing programming interaction objects, and the User Interface Management System made up of conceptual architectures for separation together with the techniques for expressing the dialogue .

The layers of programming development tools contribute to how human computer interaction affects the programmer.

6.0 Tutor Marked Assignment

1 (a) Explain the two elements of the windowing systems

(b)Using the pictorial representation of a windowing system, describe its role in interactive programming

1. Describe the architecture of a Windowing System using the client server architecture diagram.
2. What are the functions of the User interface toolkits?
3. In what ways are widgets used to support prototyping?
4. What is the alternative to the use of a took-kit? What could require the use of such alternative?
5. Using a suitable pictorial representation, describe the concept and implementation of the Seeheim model
6. What is the concept supporting the model 'View Controller'(MVC)? How useful is this concept to the programmer?
7. Explain how closely is the relationship between the MVC model and the Presentation, Abstraction and Control (PAC) model?
8. In what form can the implementation of the User Interface Management System be carried out?

7.0 Further Readings / References

- Buxton, W., *et al.* "Towards a Comprehensive User Interface Management System," in *Proceedings SIGGRAPH'83: Computer Graphics.* 198 3. Detroit, Mich. 17. pp. 35-42.
- Kasik, D.J. "A User Interface Management System," in *Proceedings S IGGRAPH'82: Computer Graphics.* 1982. Boston, MA. 16. pp. 99-106.
- Scheifler, R.W. and Gettys, J., "The X Window System." *ACM Transactions on Graphics*, 1986. 5(2): pp. 79-109.
- Myers, B.A., "User Interface Software Tools." *ACM Transactions on Computer Human Interaction*, 1995. 2(1): pp. 64-10 3.

  Palay, A.J., *et al.* "The Andrew Toolkit - An Overview," in *Proceedings Winter Usenix Technical Conference.* 1988. Dallas, Tex. pp.