USER FRIENDLY INTERFACE TO THE UNIX SYSTEM

PROJECT REPORT

SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENT
FOR THE AWARD OF DEGREE OF

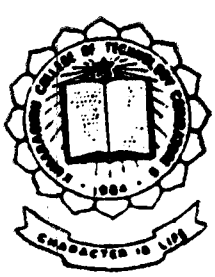
BACHELOR OF ENGINEERING IN COMPUTER TECHNOLOGY & INFORMATICS

OF THE BHARATHIYAR UNIVERSITY
COIMBATORE-641 046

SUBMITTED BY
Kochugopan. M
Bhashyam Srinivas
Reni Samuel
Satish. R

UNDER THE GUIDANCE OF

Mr. M. SENTHILKUMAR, B.E. M.I.S.T.E.



Department of Computer Technology & Informatics
KUMARAGURU COLLEGE OF TECHNOLOGY
COIMBATORE - 641 006

1989 - 90

Department of
Computer Technology & Informatics
KUMARAGURU COLLEGE OF TECHNOLOGY
COIMBATORE - 641 006

CERTIFICATE

This is to certify that the project report entitled

USER FRIENDLY INTERFACE TO THE UNIX SYSTEM

Has been submitted by

Mr. .....

in partial fulfilment for the award of Bachelor of Engineering in the

COMPUTER TECHNOLOGY AND INFORMATICS BRANCH of

BHARATHIAR UNIVERSITY , COIMBATORE during the academic year

1989 – 90.

Guide

Head of the Dept.

Certified that the candidate was examined by us in the Project work viva – voce examination held on ..... and the university register number was .....

Internal Examiner

External Examiner

Dedicated to novice users of UNIX

Acknowledgement

ACKNOWLEDGEMENT

With so many people to thank, it is impossible to thank them all. A few however, needs mentioning.

We are grateful to our Principal Prof. R. PALANIVELU,B.E.,Msc(Engg),F.I.E.(India),M.I.S.T.E.,M.I.S.T.R.S. for the ample facilities provided to us for carrying out the Project.

It is indeed a matter of great pleasure to thank Head of our Department, Prof. P. SHANMUGHAM, B.E., MSc(Engg), M.S. (Hawaii), M.I.E.E., M.I.S.T.E., who extended us kind co-operation and continued support during the course of our Project work.

Our profound thanks also goes to Mr.M.SENTHIL KUMAR, B.E., M.I.S.T.E., Associate Lecturer of our department for his valuable guidance.

We would be failing in our duty, if we do not thank Mrs. Paranjothi, M.E., Lecturer, Department of Electronics and Communication Engg, PSG college of Technology, Coimbatore who helped us by providing constructive suggestions.

Last, but not the least, we would like to thank Ms.R.PAVAI, B.E., M.I.S.T.E., Assistant Lecturer and other staff memebrs of our department who helped us directly or indirectly resulting in the completion of our Project.

Team Members.

Synopsis

SYNOPSIS

The aim of the project is to design and implement an user-friendly interface to the UNIX system. The interface seeks to make a novice user understand the intricacies of UNIX environment through an interactive, menu-driven conversation.

The success or failure of an application often rests on how easily the application can be used. The operating system stands for the interaction between the user & machine. A shell (or the command interpreter) interacts with the end-user. The shells currently available for UNIX systems are not friendly. Hence an interactive interface is necessory to make the novice user get acquainted with the commands. This interface will act as a buffer between UNIX and the new user, allowing the user to have a continuing dynamic dialogue with the machine.

The interface designing has been achieved through a functional classfication of the complete command set which enables the user to easily locate the command. The user is invited for a menu-driven conversation with the machine. Interactive session has been programmed using shell scripts and C language has been used to draw menus.

The report also describes the approach to command set partitioning together with the concepts of advanced graphical user interfaces.

Contents

CONTENTS

1. INTRODUCTION TO THE UNIX OPERATING SYSTEM

1.1 INTRODUCTION

UNIX is the Rolls-Royce of Computer operating systems, primarly intended for program developemnt and document preparation environments. The minimum configuration neeaed to support a typical multiuser time-sharing system is 16K bit processor with 256kb of main memory and fast access drive. It is written in a high level language, C with careful isolation and confinement of machine-dependent routines, so that they can be fairly easily ported to different computer systems. As a result, versions of UNIX are available for personal computers, microprocessor based systems, minicomputers and midicomputers and large main frames and supercomputers. The source code level compatibility of programs developed under Unix makes it possible to transfer applications between Personal Computers and larger machines, such as the IBM 370, Univac and even Cray and Amdhal machines.

HISTORY:

During 1965-1969, Bell laboratories participated with General Electric (now Honeywell) and Project MAC at the M I T, U.S.A in the development of Multics systems. In 1969 BELL laboratories withdrew from the Multics effort. Some members of the research staff under Ken Thompson began to work on developing a system which was later christened UNIX. It was a single user system for the PDP-7 family of computers written

in Assembly language. In 1972 Dennis Ritchie also joined the team and the entire OS was written in C. This made the availability of UNIX to users outside of AT & T.

The following years saw UNIX springing up in different versions. The most popular of them are given below:

1. Standard UNIX system from AT & T
2. XENIX, the microcomputer version of UNIX from Microsoft
3. AIX, from IBM
4. Ultrix, to run on DEC machines
5. A/UX, the one from Apple
6. Berkely Version BSD4.2 from University of California at Berkeley

1.2 FEATURES OF UNIX:

UNIX opens up the world of minicomputer and mainframe, computing to 16-bit microcomputers. It is a powerful operating system that brings multi-tasking, a reportoire of system commands, and an extensive set of system libraries to 16-bit microcomputers.

The features of UNIX system include

* Multitasking capability
* Multiuser capability
* Transportability
* Large section of powerful UNIX supplied programs
* Communication and Electronic mail
* Library of applications software

1.3 THE STRUCTURE OF THE UNIX SYSTEM:

Unix has some stuctural similarities with single user microcomputer operating system like CP/M and MS-DOS in that it has a central program that remains in memory at all times and a command interpreter that can be replaced by an application program or other system utilities like editors, compilers when they are invoked. In Unix, the central program is called the kernel and the commmand interpreter is called a shell. Commands can be built into the command interpreter or contained in System files. However, both the shell commands and file commands that come with Unix are much more extensive.

THE KERNEL:

As its name implies, the kernel of Unix is the central program of the operating system. It consists of collection of routines and data structures that are permanently housed in the computer's main memory and perform lower level tasks, such as transferring data between the computer and its peripheral devices.

The entry points to the Unix are handled by interrupts. The interrupt is an event that causes the computer to stop what it is doing and perform some special processing task. So, the kernel can be thought of as an event-driven or interrupt-driven program. The essential features of the Unix operating system can be highlighted under the following needs:

(a) Hierarchial File System:

Unix provides an "acyclicgraph" structure for its file system with the directories forming the nodes. Starting at the "root" of the file system, the file system is further divided

into directories and files. Each user is assigned a "home" directory, where he can create his own sub-diretories and files. The user can traverse the file system tree subject to the access right.

(b) Unix tools:

Unix has variety of commands called tools for achieving various tasks. The user can edit a report or a program, send an electronic mail, or check spelling mistakes in his documents. Each tool does a specified job, which can be combined together by suitable means to construct new commands. Unix programmers can use these tools to good advantage to edit, compile, debug and manage their program development in the C programming language.

(c) Input-Output:

Effective processing of text is an important central goal of UNIX. A program to process text is called a filter. In Unix, a filter is a program that accepts input from standard input and sends its ouput to the standard output. The default source for standard input is the keyboard, and the default destination for the standard output is the screen. Unix treats devices such as Keyboards, screens and printers as files. I/O redirection boils down to the ability to control the flow of a program input and output to and from any specified file by operators, "<" & ">". Also the standard O/P of one program can be made the standard input for another program by pipe operator "|". This helps in connecting together commands to make complex commands.

(d) Shell:

Unix provides a powerful command language, the "shell". Unix shell is really a command interpreter which reads and executes operating system commands written in a shell language. The shell also supports parameter passing facilities, so that complex sequences of commands can be grouped together into a shell program and executed. It provides, through the shell variables, an appropriate environment to work with. Each shell has its own lanugage for these commands. Two major shells come with UNIX the Bourne or standard shell 'sh' and the C-shell 'csh', developed at the university of California, Berkeley.

1.4 PROBLEMS WITH UNIX:

Though Unix operating system is easy to use and get the work done, it is difficult for a beginner to understand where to start and to know how to get the required work done on the system. It is difficult for the user to know the best use of the facilities available on the system. Norman says that the system design is elegant, but the user interface is not. Unix shells are not user friendly, which can be mainly attibuted to two main reasons:

1. The large number of Unix commands and the inconsistency among them.

2. The number of options available for each of the commands and their meaning with different options.

The problems with Unix is that, it fails in several simple checks like the following:

a) Consistency:

UNIX command names are not consistent, the pattern of naming varies from command to command. While the "data" command name is kept in its totality, "remove" is abbreviated to 'rm' and a pattern matching command to 'grep'. The commands take many special argument flags and the manner in which the flags are specified is not consistent, varying from commmand to command.

b) Functionality:

The command names, formats, and at times, the syntax seem to have no relation to their functions. UNIX uses abbreviation even when the original name is already sufficiently short. For example, "user" is abbreviated to 'usr'.

c) Friendliness:

UNIX is hidden from the user and silent in operation. The user has no feedback on the job done by the systm. These are more than 300 commands, one corresponding to each of the application tools and utilities available on the UNIX system. A beginner does not know what commands he need to learn, to begin with. Even, if a few commands are named for him he will not be able to remember them all easily, as their names do not sound related to the jobs they do. As, there is nc enough feed

back on the job done by the system, even a slight ignorance on his part can put him into danager and do irreparable damage to him.

1.5 NEED FOR A FRIENDLY INTERFACE:

Learning on UNIX is not easy because of the complexity of the commands and at times the inappropriateness of their names. The following example will support the above statement. The Unix command to list a directory is 'ls'. Bell version 7 offers 11 options which includes l,t,a,s,d,r,u,c,i,f and g. In addition to this Bell system 3 offers o and g, system 5 offers p & the Berkeley gives m,l,C,g,x,F and R. Each option stands for some particular meaning. Similarly the UNIX command for a searching for a particular pattern in a file is 'grep' which is quite inappropriate. Shneiderman mentions that a large number of functions provided by an operating system results in slower learning on part of the user, increased number of errors, and frequent recourse to referring manuals. Hence there arises the need for some help to the user in understanding and using UNIX.

When the need for the user friendly, Interface is discussed, a word should be mentioned about the online documentation provided by Unix system. Most Unix systems maintains on-line documentation about each command, but it is difficult to find such documentation on smaller microcomputer implementations with limited disk space (for eg: with 10

megabytes or less of disk space). Unix systems provide a tool called "man" in order to assist the user in gaining familiarity with the system. But this, however requires the user to know the name and type of the command. This serves as an on-line reference to the specified command but does not serve the purpose of an interactive assistance during a typical UNIX server.

Dr.Rebecca Thomas [1] has pointed out the difficulty of referring hard copy manuals when one is working with system V. The two books UNIX system users manual and UNIX system administration manual are cumbersome to go through because of their bulky size.

Bell labs has recognised the need for a friendly interface and they believe that beginning and infrequent users would be even better served by a more helpful interface, such a menu driven shell that would spare them the need to remember all operation details from one session to next.

Of late, the AT & T system V-version 4.0 has been released with a better outlook. They have beautified the unfriendly look of UNIX by implementing a graphical interface - the Open Look. Appendix includes a copy of their ad.

2. SHELL PROGRAMMING CONCEPTS

As it is, it is a programming language in which each statement runs a command. It must satisfy both the interactive and programming aspects of command execution. Program written in the shell or another language maybe helpful enough that people would like to use it. Major theme in shell programming is making programs robust so that they handle improper input and give helpful information when things go wrong. The common use of a shell program is to enhance the user interface to a program.

2.1 SHELL VARIABLES AND COMMANDS:

Variables offers symbolic way to represent and manipulate data.

Environment variables:

Each process has its own environment which is a list of string variables that is passed along with any command parameters. A process can access these variables via address passed to it as arguments for its main program. These variable contain useful information about the user to whom that particular process belongs such as home directory, starting shell and path for searching for commands.

Shell Variables:

Each shell can have a set of variables distinct from its environment and are stored as program variables within the shell. Shell variables may be those created and maintained by

the UNIX system itself or by the user. To create a shell variable type something of the form $ Name = Value. To see the system set variables type

$ Set

A local shell variables known only to the shell that created. It can be made global by using the export commana.

Eg.: Export cat

Moreover there are commands that are part of shell program itself.

Eg.: Set, :, ., Export & Shift

2.2 SHELL SCRIPTS:

Shell scripts are created by placing UNIX commands into text file. Features that can be used to elevate the powers of shell scripts are:

1. Shell scripts can take arguments that can be symbolically represented within the script just like most commands.

2. Shell allows you to create shell variables that assign values to them.

3. The shell has a collectaion of predefined metacharacters, characters, assign special means that greatly expand the applicability of shell scripts.

4. Shell provides control structures such as if...else constructs and for loops that let you implement programming language approaches.

A script can be designed so that it can except additional information at run time. The first approach is to write an interactive script that requests the information it needs from the user. The second approach is to have the script except arguments from the command line.

2.3 ERROR CHECKING:

The idea is to head off errors from the part of the user. The first step is to anticipate possible errors that the user might make. The next step is to device a way to detect each error. The test command and the case matching abilities are useful for this. After an error is found, something should be done about the error. If and case statements are useful in organising possible actions. A non-interacting program may give an error message and quit where the EXIT command is useful. An interactive program may give the user a chance to correct the error.

The following example shell scripts will illustrate the error checking feature. Consider the following non-error checking example.

```
echo enter the file name you wish to edit
read file
Cat $ file
```


The shell facilities makes it possible to create
the form of shell scripts.

prepared to test them. Second important point is to make your scripts compatible with the rest of the UNIX system. The third, it is important to know the strength and weaknesses of shell programming approach.

a) Troubleshooting:

The most important troubleshooting tool offered by a system is the combined -vx option of the shell, which lets you trace the execution of a script step by step showing you what substitutions the program makes as it goes along.

Then we can prepare specially tailored files or arguments to test a program. Another step is to place temporary files in your working directory and not to have the script, to remove them during development so that you can check their contents. Similarly strategically placed echo statements can help you track down what is happening.

b) Compatibility:

A shell script intended for wide spread use should incorporate the following features in order to be compatible With the other system programs:

1. Error Checking
2. Error messages directed to the std error
3. Non zero exit status for abnormally terminated programs.
4. Options if any to be indicated by hyphens and to come between the procedure name and the other arguments
5. Cleaning up after normal or abnormal termination

3. UNIX COMMAND SET PARTITIONING

3.1 INTRODUCTION:

UNIX provides a variety of commands for various tasks. But because of the difficulty in gaining familiarity with a particular command only a few of them seem to be frequently used. It has been observed that out of a total of more than 300 commands 10% of the commands account for 90% of the command usage.

The reasons as to why there ae wide-disparities in the frequencies of command usage are: redundancy among commands, difficulty of use and the infrequent use of them. Moreover, on many computer systems all the commands are put at a place, the unused commands seem to as prominent as the set of frequently used commands. This is likely to intimidate new users, hinder the learning of commands set, make identification of important commands difficult and lead to erroneous command entry.

Besides this there seem to be sets of inter-related commands that go together. Eg., when a user changes a directory with the 'cd' command, he is likely to check where is in the directory hierarchy with the 'pwd' command. Hence the commands need to be grouped together into some major categories, depending primarily on the functions they serve.

HANSEN et.al. classified the commands into five major classes based on the functions they perform and frequency of use.

1. GENERAL EDITING COMMANDS:

The most frequently used commands includes both text editiors that manipulate text and stand alone UNIX programs that perform actions on non-text objects such as files and directories.

2. ORIENTING COMMANDS:

These commands inform the user's about the environemnt in which they are working and the status of the objects and the processes in that environment.

3. PROCESS MANAGEMENT COMMANDS:

These commands used to integrate individual commands into complex units (pipe) to provide parallel execution (&) and to terminate process (kill)etc.

4. SOCIAL COMMANDS:

These commands allow people to exchange information with others.

5. TASK SPECIFIC COMMANDS:

These are required for achieving specific tasks like document preparation.

These commands can also be grouped depending on the context in which they appear some commands may either precede or succeed a variety of other commands.

For the purpose of easiness of description designers incorporate their own style in classifying commands. William.B.Twitty(5) has summarised UNIX commands into twelve partitions such as (1) Editor commands (2) directory commands (3) Mail and user communications (4) File manipulations (5) Miscellaneous utilities etc. However it is left to the choice of designer to follow a particular convention.

3.2 USFUNIX PARITIONING:

In our interface (Christened USFUNIX) the commands are classified into functionally similar categories. Commands that do some inter-related types of tasks are grouped under one head. Each head is further sub-divided until an indiviaual command is reached. Commands whose functions overlap may appear under more than one head. All commands on the system are split into a tree like structure.

To start with, since the UNIX system distinguish between the super-user(system admiistrator) and an ordinary user, the commands are divided into two partitions - the system partition and user partition. The former contains commands related to the system administration and maintainence. The latter contains commands which are used by an ordinary user in working with the system. These partitions are further sub-divided.

SYSTEM PARTITION:

These are further divided into two groups, system Accounting and system maintenance. The former group contains commands for system accounting used for accounting of files and system activities. The latter group contains commands for tape and floppy backup and recovery operation on i-node system checks and process related commands including system intializtion.

USER PARTITION:

The user partition has been broadly divided into 10 heads. Commands have been allocated into groups depending upon their functions. Each of these heads are further sub-divided to make it easier for the user to selet the commands he wishes.

1. FILE & DIRECTORY MANAGEMENT:

These commands are involved in manipulating files and their contents, working on directories, file storages, file comparision and file security.

2. TEXT PROCESSING:

These are used in creating and manipulating text and include both screeen based systems like the line editor and visual editor.

3. INFORMATION HANDLING:

These commands are used for searching and sorting files, selecting data from files etc.

4. LANGUAGE PROCESSORS:

These includes compilers for FORTRAN, C and PASCAL etc.

5. SOFTWARE DEVELOPMENT:

These are involved in the development and maintainence of software and include debugging, source code control systems and code checking commands.

6. SYSTEM DOCUMENTATION:

These include commands like man learn etc, for assisting the user about the system.

7. OUTPUT OPERATIONS:

These are used for printing, and for storage on disk, floppy and tape.

8. COMMAND EXECUTION:

These are used in the execution of commands to repeatitively execute a command changing the arguments read/write data, to know the re-execution time of a command etc.

9. COMMUNICATION:

This section contains commands needed for communication with other user's and systems.

10. MISCELLANEOUS UTILITIES:

This section contains some most frequently used utilities and tools to manage the system data and processes.

Table 1 gives the command groupings and Table 2 provides the individual tree structured menus. A complete command tree structure can however be seen in Appendix A.

TABLE 1

COMMAND GROUPINGS	FIGURE REFERENCES
* SYSTEM PARTITION	Fig. 1
# SYSTEM ACCOUNTING	
# SYSTEM MAINTENANCE	
- USER COMMUNICATION	
- BACKUP	
- FILE CONTROL	
- SYSTEM CONTROL	
- PRINTER CONTROL	
* USER PARTITION	
# FILE AND DIRECTORY HANDLING	Fig. 2
- DIRECTORY MANAGEMENT	
- FILE STORAGE	
- FILE COMPARISON	
- FILE SECURITY	
- FILE OPERATIONS	
> FILE CONTENTS	
> FILE COMPRESSION	
> FILE MANIPULATIONS	
# INFORMATION HANDLING	Fig. 2
- SORTING	
- SEARCHING	
- FILE MANIPULATIONS	

Fig. 3

LANGUAGE PROCESSORS

- C LANGUAGE
- FORTRAN
- ALGORITHMIC LANGUAGE
- MACRO PROCESSING
- COMPILER GENERATORS

Fig. 3

SYSTEM DOCUMENTATION

Fig. 4

SOFTWARE DEVELOPMENT

- SOURCE CODE CONTROL SYSTEM
- PROGRAM DEVELOPMENT

Fig. 4

COMMUNICATIONS

- CALENDAR
- MESSAGE
- TERMINAL COMMUNICATION

Fig. 5

OUTPUT OPERATIONS

- PRINTING
- STORAGE
  - > FLOPPY
  - > TAPE
  - > DISK

Fig. 5

COMMAND EXECUTION

- PROCESS EXECUTION
- STATUS ENQUIRIES

Fig. 6

TEXT PROCESSING

- DOCUMENT PREPARATION
- DOCUMENT FORMATTING

Fig. 5

MISCELLANEOUS UTILITIES

- GRAPHICS
- GAMES
- NOVELTIES

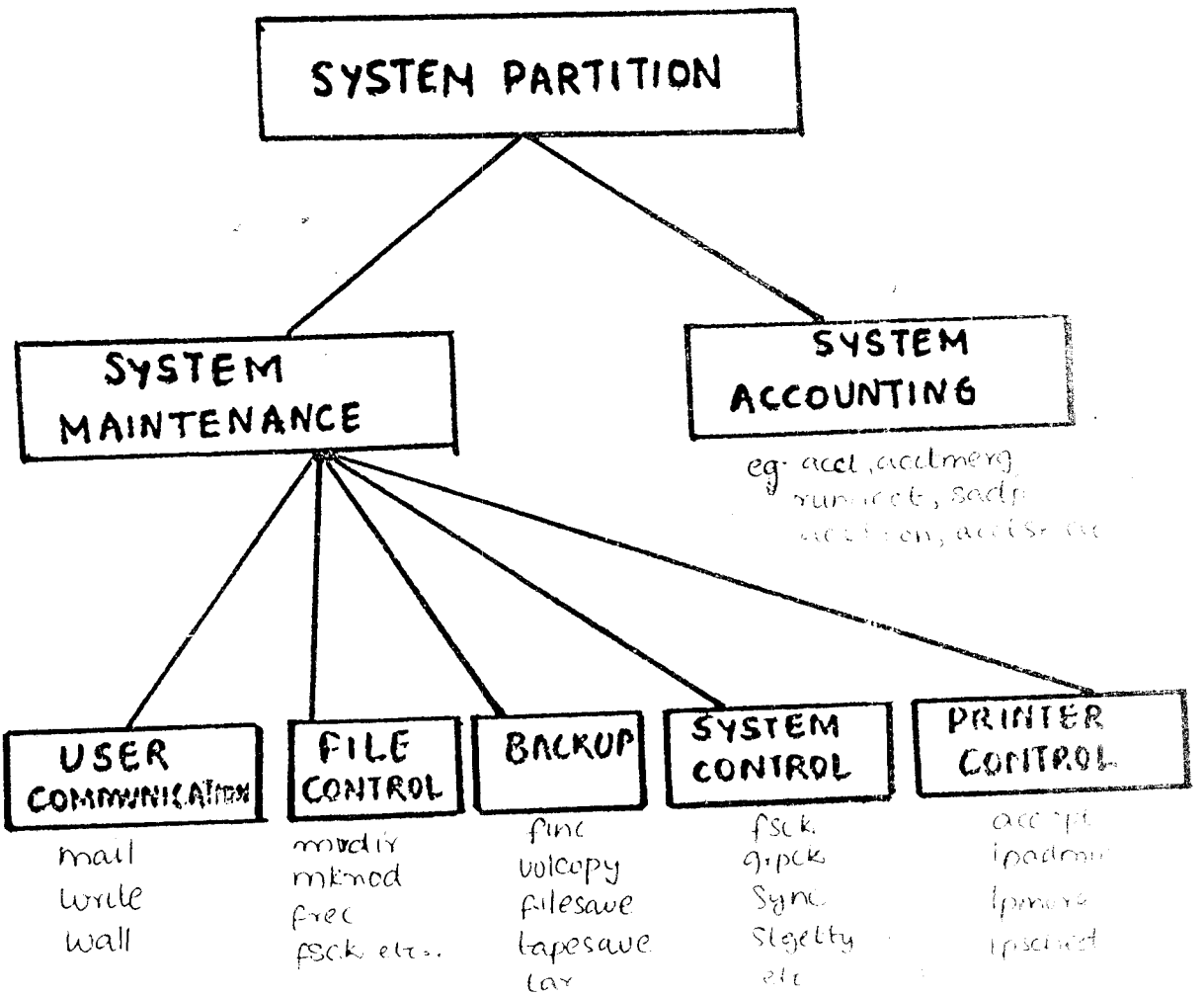


Fig 1. Partition Listing (partial set only)
(System Partition)

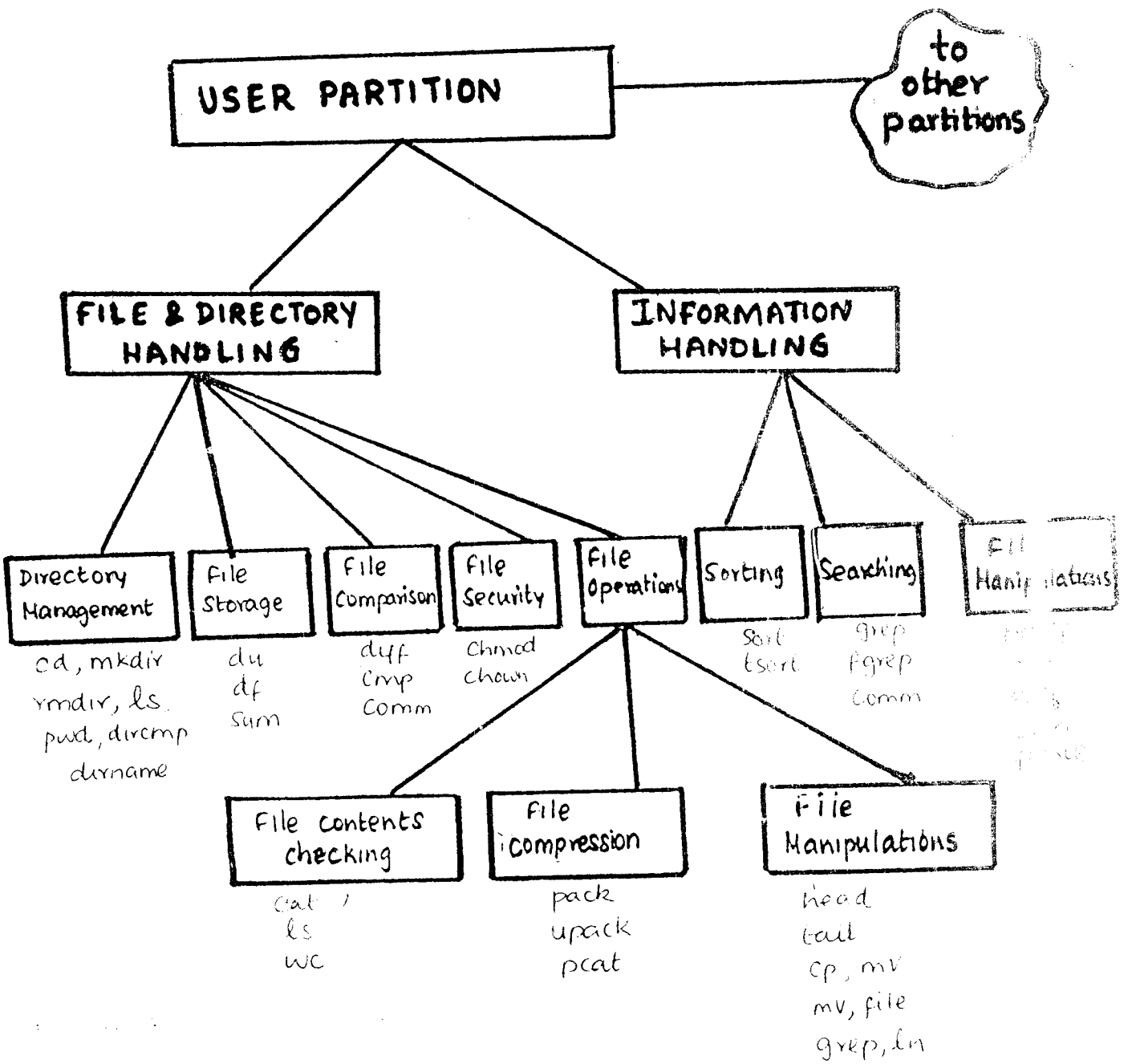


Fig 2. Partial listing showing File & Information handling

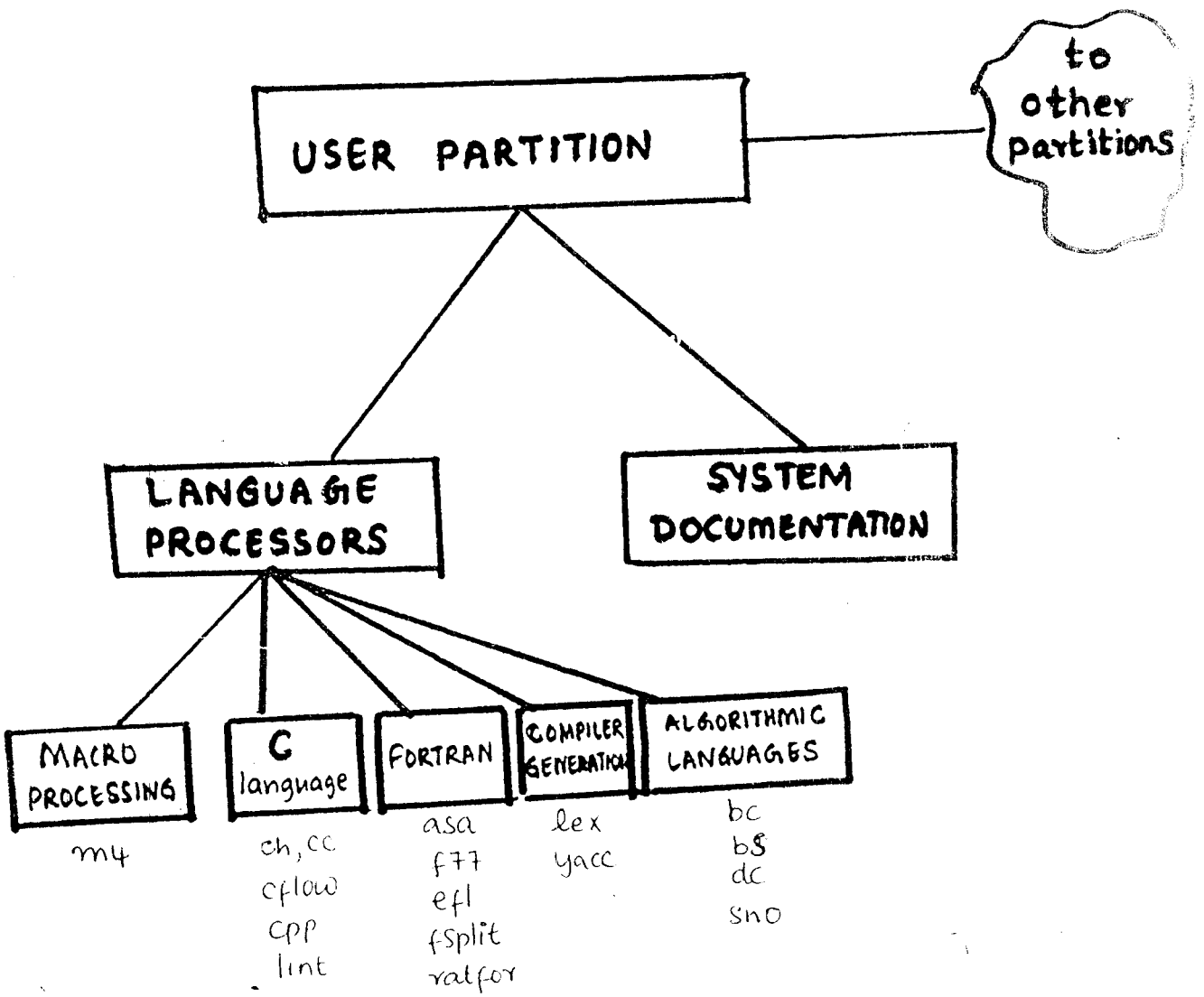


Fig 3. Partition listing showing language processors and System documentation

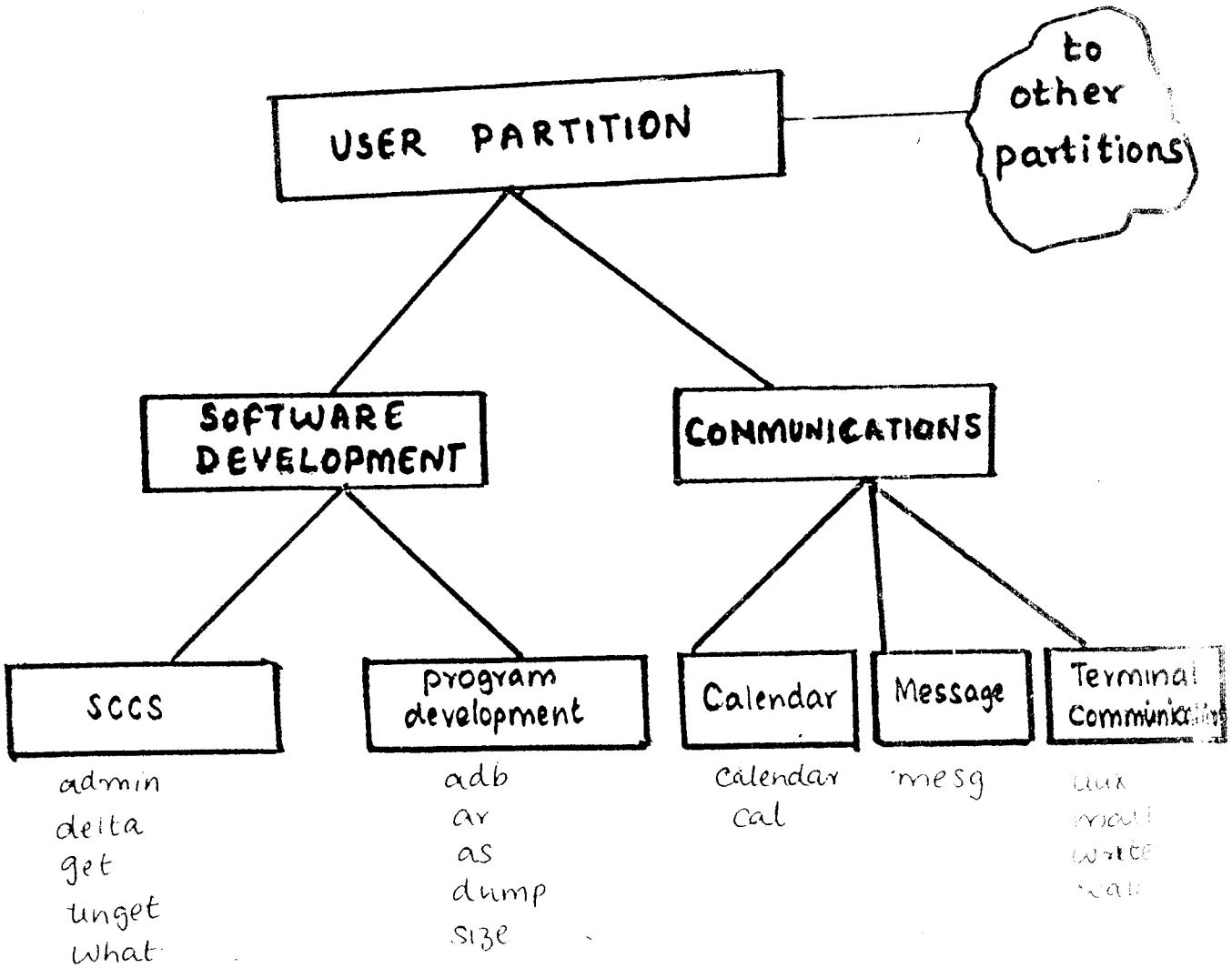


Fig 4. Partition listing showing Software development and communications partitions.

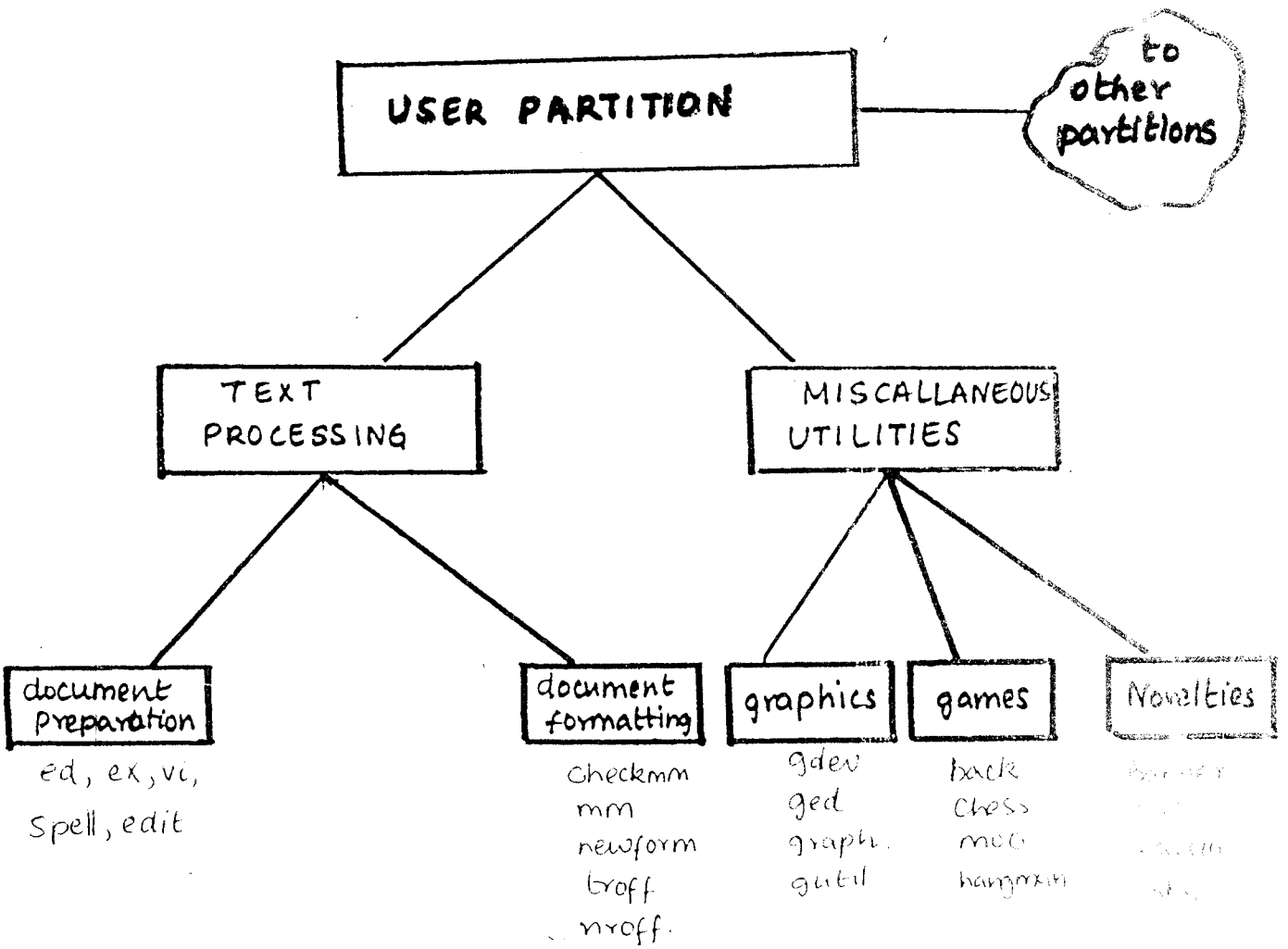
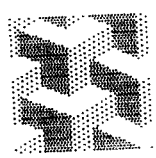


Fig 6. Partition showing Text processing and Misc. Utilities partition



Design and Implementation

4. DESIGN AND IMPLEMENTATION

4.1 GENERAL IMPLICATIONS:

Providing an easy-to-use "human interface" for users is an increasingly important requirement for operating systems. Such a connection between machine and the human that use it plays an important role in the overall productivity of the system. The design of an interface involves in the evaluation of the human-computer interaction. This involves in understanding the cognitive and perceptual abilities of the users. The users abilities to recognize changes in displays occuring before him and initiating appropriate action should be taken into consideration. Concerns about perception include response times, reading text, color vision, sensitivity to flicker and contrast and visual fatigue. Many psychological factors like anxiety, tolerance for ambiguity and motivation are to be considered. The consciousness of the novice users about the computer system also needs to be taken into account. They are assumed to have no knowledge about using the system and probably little knowledge about the computer issues. They may arrive with anxiety, which inhibits leaving about using the computer. These limitation have to be overcome by the designer.

A novice user should be asked to carry out only simple tasks to build confidence and reduce anxiety in him. He wants more informative feedback to confirm his actions.

Effort should be made to attract the attention of the user with respect to exceptional condition. This can be done by a variety of ways-marking, blinking, color etc. However the displays should be simple and logically organised, and clustered displays should be avoided. In UNIX we have two header files namely, 'curses.h' and 'terminfo.h' which can be used for this purpose.

4.2 DESIGN GOALS:

a) Functionality:

This involves in ascertaining what tasks and subtasks are to be arried out and paying attention to usage patterns. Systems with inadequate functionality frustrate the user and are often rejected or underutilised. However, excessive functionality can make implementation, maintenance, learning and usage more difficult because of the complexity involved. Another issue is to compatability with other systems.

The design and implementation convenience should not dictate system functionality or command features. Higher level tasks can be broken down into lower level atomic tasks that the user executes with a single command or a menu selection. However, if the atomic actions are too small, the user will become frustrated by the large number of actions necessary to accomplish an higher level task. Complex interactions between its components give the program most of its power, but at the same time they present a formidable obstacle to understanding and extending it.

b) Reliability, Consistency and Interval Rigidity:

Since the specification and therefore the module structuring is considered fixed, one of the most effective methods for enforcing it is the use of redundant descriptions and consistency checking which will ensure proper reliability. Attention must be paid to correct performance integrity and protection against unwarranted access, inadvertant destruction of data etc. Consistant actions should be required in similar situations and identical terminology in dialogues and prompts. In a well executed conventional implementation projects a great deal of internal regidity is built into the system, ensuring its orderly developement.

c) Organisation of Interface:

This should be done keeping the functionality, the purpose which the interface should serve, in mind. The items to be presented to the user should be classified baed on their functions. This helps in providing easy access to and visibility of the items. One technique heavily employed throughout the interface design is the use of menus. A menu is the type of window that causes a specified operation to be performed when a selection is made in that window. This also help the user in formating a good mental image of the whole structure more easily and quickly. We have made use of the features of the header file 'curses.h' for this purpose.

d) Feedback:

Informative feedback about the accomplishment of each task is helpful and constructive. Specific error messages should be provided when errors occur. Through for minor actions little or no feedback may be provided, the infrequent and major actions should be provided with a more substantial feedback. Specially in relation to commands that destroy data, it is desirable that the user be remained of his action and be carried out only after an explicit response.

e) Errors:

Normally the use of an unavailable feature results in an error. The interface should primarily aim at reducing the error rate. The user should be provided appropriate feedback on the errors committed by him and with the corrective measures suggested for general approach should be used so as to make learning easy. The learner, on the other hand, has a powerful and reasonable, curiosity to find out what happens when he does something wrong. A system must protect itself from all such errors as far as possible, protect the user from any serious consequenes. The system should be engineered to make catastrophic errors difficult and to permit recovery from as many errors as possible.

f) Error Messages:

Providing good error messages serve as an invaluable training and to the learner and as a gentle reminder to the expert. With a grahic display, it is possible to present error

messages rapidly without working the user's time. Error messages should be specific, indicating the type of error and the exact location of the error in the text. An intelligent error handling scheme that helps to correct errors rather than compound them, can be of great help. It is desirable that the user be indicated various alternatives and a provision to correct and reexecute an erroneous command line.

g) Help Line:

Help line in each menu provides a ready reference to the usage of the interface. For knowledgeables, but intermittent users, who maintain the semantic knowledge of the task, but have difficulty in maintaining the syntactic knowledge on line help screens will help them in filling the missing pieces of knowledge.

4.3 DESIGN OF INTERFACE:

a) Design Consideration:

The USFUNIX interface has been designed keeping the novice user in mind. It aims at assisting the user in comprehending and effectively using the large number of UNIX commands. The interface is basically menu-driven whereby the user enters the appropriate responses as requested by the menu. Main program of the interace has been written in the "shell" command language where as menu display programs are written in 'C' lanugage.

The design basically involves in:

1. Command set partitioning
2. Design of menu
3. Writing interactive routines for each command

The shell command language has been used for programming most of the interface. This command language has been preferred to the conventional programming language because it offers:

1. A very high level language approach
2. Simplicity in usage
3. Immediacy of results
4. A congenial environment

It is always desirable to use built in facilities available in a system. Making use of system calls and built in utilities, menu display part of this interface is written using C language. A programmer can take advantage of existing software and try to write the minimum amount of code to accomplish the job. The results are moe uniform and easier to understand and maintain for the designer.

b) Menu Arrangment:

The main-menu incorporates the system partition and the user partition. Provision is given to view the help file describing the structure of interface. The system partition leads to a hierarchial tree structured menu which contains

commands for Maintenance and Accounting of the system. Similarly the user partition consist of commands that are more related to the user.

The user-partition leads to set of tree structured menus which incorporates the different partitions as follows:

1. File and directory handling
2. Text Processing
3. Information handling
4. Language processors
5. Software development
6. System documentation
7. Output operations
8. Command Execution
9. Communications
10. Miscellaneous utilities.

User is led through a set of menus until he reaches the command menu. On selecting the necessary commands from the command menu the user is provided information about the particular command which he/she selects and the user is prompted for the following menu.

Please enter your choice:

1. For USFUNIX mode
2. For viewing manual help
3. For quitting this command

On selecting 1 the shell script for that particular command is called and then the user is led to a conversation with the interactive interface. On selecting 2, the system documentation of the particular command is obtained. This gives the complete details about the command together with all the available options. Selecting 3 takes him back to command menu.

Similary the system partition is mainly of two fractions - the system accounting and system maintenance. The user has to go through a set of menus as above to reach the command.

4.4 IMPLEMENTATION:

The shell programming has been done under Bourne Shell. The menus has been drawn using C. The software has been successfully tested on a NEXUS 3500 machine running UNIX. We have infact used multiple windowing feature of NEXUS 3500. The novice user can press F1 during any session and then a new pad will be opened. He can type the command at the shell which he had learned and have a trial of executing commands from the shell. Ofcourse, it then requires the user to abort the pad and come back.

Summary and Suggestions

5. SUMMARY AND SUGGESTIONS

It has been quite a rewarding experience for all of us in having what we have done until now. But the scope of our work and the paucity of time have prevented us from going deeper into the subject.

USFUNIX, with its menu driven structures guides the user as to how it should be used providing on-line help when necessary. It thus serves the dual function of an interface to the end user and an on line documentation.

When we started our project, the version of UNIX from AT & T was version 3. However in March '90, AT &T has come up with a modification in their UNIX. They have brought up the version 4 with a graphical interface - The open look. This has however turned the unfriendly looking Unix into a more friendly environment.

Our project also can be modified to make the environment look more graphical. The user can move the input device (cursor or mouse) and select the particular item from the menu. Further modifications can lead to an integrated environment wherein all the menus could be incorporated in a single screen and the facility of popping up menus can be used.

We also feel that such a Interface (like open look) would surely come up to the expectations of the novice user and thus make him feel more comfortable with the environment. More sophisticated interfaces which may sprang up in the future may incorporate natural language techniques which makes the user feel more comfortable with the environment.

Bibliography

6. BIBILOGRAPHY

1. "An user guide to the UNIX system" , Dr. Rebecca Thomas, Jean Yates, Osborne - Mcgrawhill, 2nd Edition.

2. "Advanced UNIX - A Programmer's guide", Stephen Prata, B.P.B. Publications.

3. "Inside XENIX", Christopher.L.Morgan, BPB Publication.

4. "The Unix shell programming Language" - Rod Manis, Marc. H. Mayer, Horward.W.Sams &'Co.

5. "Unix on the IBM PC", William.B.Twitty, BPB Publications.

6. "Unix Programming on the 80286/80386", Alan Deikman, BPB Publications.

7. "Unix Utilities - A programmer's Reference" R.S.Tare, Mcgraw-Hill Edition.

8. "Advanced Programmer's guide to UNIX system V" - Rebecca Thomas, Lawrence R.Rogers, Jean L.Yates, McGraw-Hill Edition.

9. "The Unix Programming Environment", Brian. W.Kernighan, Rob Pike, Prentice Hall of India (P) Ltd.

10. The ALTOS system V 386 Operating System, system information guide, ALTOS computer systems, USA.

11. UNIX System V manuals, AT & T, USA

12. NEXUS 3500 Manuals, Apollo Computer Inc, USA

13. "Dataquest", May 90, (Page 94).

14. "The C Programming Language", Brian.W.Kerninghan, Dennis.M.Ritchie, Prentice Hall of India Ltd.

15. "Advanced C:Techniques and applications", Gerald. E. Sobelmann, David. E. Krekelberg, Que Corporation. pp 145-157.

Appendix

A. Complete Command set partitioning

APPENDIX B

ADVANCED GRAPHICAL USER INTERFACES

INTRODUCTION:

The success or failure of an application often rests on how easily the application can be used. For instance, any application that requires a great deal of memorization is likely to scare away users, and the slight of a two inch thick user's manual is enough to discourage even an experienced programmer. This disouragement however can be prevented by a clear presentation of the applications to users.

Creating an applications that is user-friendly often requires more fore thought and planning than creating the application itself. However, if a general-purpose user interface is developed and the functions are placed in a library, the amount of work required to implement a user-friendly application program is reduced greatly.

THE USER INTERFACE:

An user interface acts as a buffer between an application program and the user, allowing an user to customize the display of information according to personal preferences. User interface are inherently interactive and provide a continuing dynamic dialog between the user and the application. In this display, the user specifies actions called events. In other words, the interface is even driven, and actions on the user's part create or destroy displayed information. Most user interface have consisted of little more

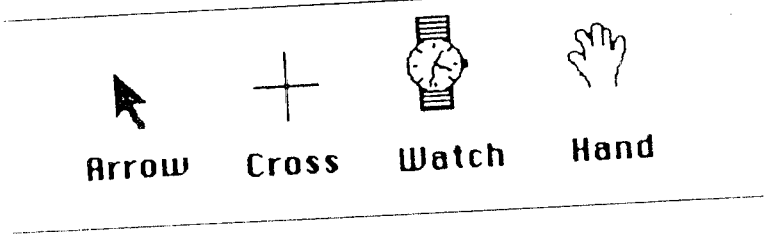


Fig a. Pointer shapes

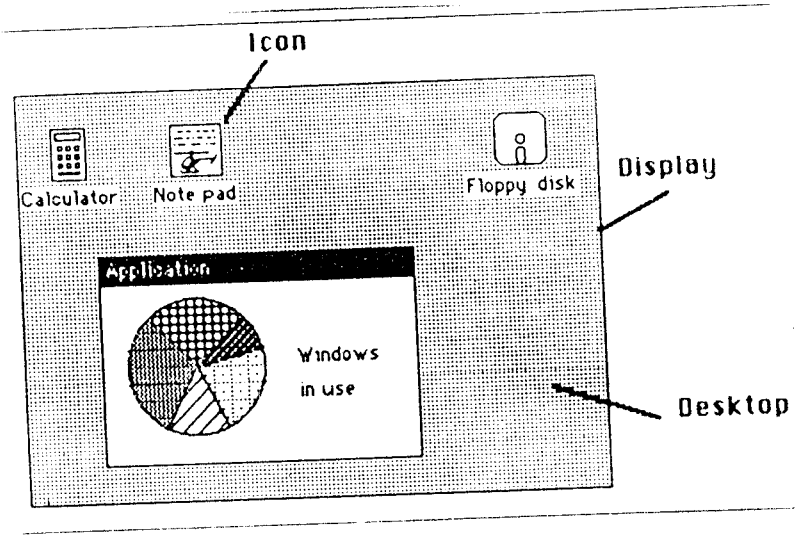


Fig b. The desktop

than a command line interpreter. The system prompts the user for input by displaying a one-character symbol. The user responds by typing textual information, followed by a return keystroke.

Contemporary user interface often use graphical metaphors of real-world objects. For example, because file cabinets are used to store important documents, a uer interface may use a picture of a file cabinet to represent a disk directory of document files. User interface that was graphical displays, most of which are of the bit-mapped variety are therefore the most flexible. With the combination of a pointing device and conventional input, we can specify virtually any type of information.

THE CURSOR:

The cursor can have many different shapes, ranging from simple arrows and crosses to more specialised symbols like human hands. (Fig a). For example, if the user is pulling or dragging, a graphical image across the display, the cursor may appear as a hand. In cases where the system is processing some information and therefore temporarily ignoring user input, the cursor may take the shape of a watch. If the user is selecting an object on the screen, the cursor may default to a simple arrow. The cursor is the place where user focus their attention because the eye follows the selection process. When the shape of the cursor changes, the user is informed immediately of a pending operation.

POINTING DEVICES:

Pointing devices include the light pen and touch screen. The light pen, when placed next to raster display, detects the exact time. When the electron beam crosses the spot at which the pen is pointing. This time is measured against an internal clock, and the peruse location of the light pen is determined. With the touch screen, we specify items with our finger. When we press our finger against the touch screen, the computer detemines the finger's location on the display. Light pens and touch screens require specialized hardware, and this requirement restricts the number of systems on which these devices can be used.

Other devices used for pointing are the track ball and the Joystick. The trackball, a small ball mounted on rollers, translates rotational movement into a change in the cursor's position. When the user rolls the ball in one direction, the cursor moves in the same direction, because the ball can pick up significant momentun, the cursor can be made to move rapidly across the screen. The joystick is similar to the track ball. When we press the control stick in one direction, the cursor moves in that direction. Both the trackball and the Joystick are useful for simple relative movements.

The most common positioning devices are the bit pad and the mouse. The bit pad consists of a hand held locating devices that moves on the surface of a specially designed

platform. Bit pads usually are designed to give precise positioning. The mouse, is an inverted hand-held trackball. Moving the mouse across any smooth surface causes an internal ball to roll, and this movement is translated into the corresponding movement of the cursor. If the mouse is picked up and placed on another part of the table, the movements continue from the new location. The mouse is relatively inexpensive and yet offers a farily high-resolution capability.

ACTIVATION BUTTONS:

Pointing devices also have activation buttons. The number of buttons varies according to the application's requirements, the type of positioning device, and the manufacturer. A single button is often best, because the user cannot press the wrong button. The process of pointing the device and pressing an activation button is called a click. For example, in a text editing application we can designate a single click for the selection of a new location for editing, a double click for the selection of a word and a triple click for the selection of entire line of text.

THE DESKTOP METAPHOR:

For the user to feel comfortable with an application the interface must be consistent. That is, when the application changes modes, the method that the application uses to communicate with the user should remain the same. For example, if we ae using the spread sheet function of a

business application and switch to a communication function, the user interface shold have identical commands for operation's common to both modes. Consistency of commands reduces the amount of memorization required for the user to operate the system and makes the program easier and more enjoyable to use. An effective, natural looking means of maintaining this inconsistency is the desktop metaphor. In this metaphor, the display shows an office desk and its tools (See Fig. b). The application functions being used also are displayed. Data elements caused icons represent the functions and choices. An icon is a small graphical object that usually includes a label. When we use the cursor to select an icon the appearance of the icon changes to signify that the selection has occured. Moving and copying the icon are straightforward. Opening an icon causes the application that the icon represents to appear on the desktop. In addition, the icon may change in appearance or disappear.

WINDOWS:

A window is a rectangle defined in world coordinates and used to select graphical data for display. The word window means a graphical element used to display data on the desktop.

By extending the desktop metaphor, we can create many useful user interface structures, such as the metaphor of paper and ink. Paper and ink are used to create documents. The ink is ordinarily black and the paper is white, and these sheets can overlap. On the computer screen, the paper is a

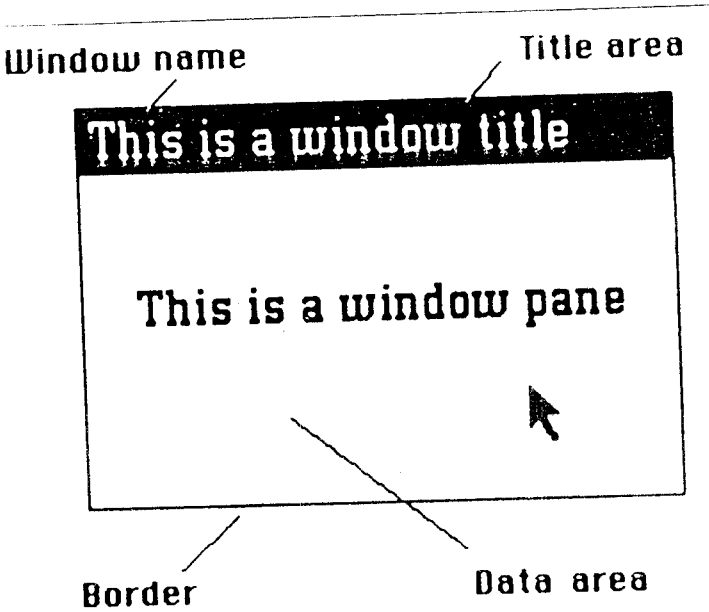


Fig c. A desktop window

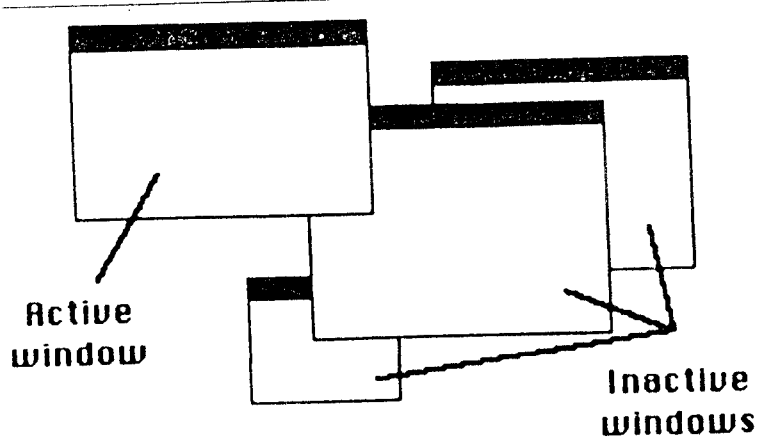


Fig d. Overlapping windows

white rectangle on which data can be drawn in black. Many rectangles can exists at once and overlap on the display (See Fig. c) Through the use of windows, an application presents information in a user-controlled format. Only one window is active at any given time. The active window is the one with which we can interact. For example, we can modify the active window's size and location on the desktop. Windows are displayed in a depth-ordered fashion. The windows overlap, and by convention, the top window is the active one. We can make any window active by placing the cursor in that window(See Fig. d).

To implement windows in a user interface, the programmer must recognize the natural limitations of this metaphor. For intance, windows must have a minimum size, which usually is detemined by the user interface. When we interface an application with one or more windows, the application must have access to the parameters of the windows and to the even occuring within the windows.

For may graphical regions of a window, save a pointer to a function that interprets the evens for that region. For example, assume that a user clicks inside the data area of a window. The user first determines the window in which the click takes place. If the click does not happen in any windows, the event is passed to a desktop function that interprets that click. If the click occurs in an inactive window, that window is "popped" to the top of the windown pile and made active. If

the click occurs in the active window, the user interface checks the window's data structure to find a pointer to the funtion that process the event. Provided that this function pointer is not NULL, the user interface calls the function with pointers to both the event and the window as parameters.

MENUS:

The standard method to control the user interface and its application is through a menu. Menus contain logically related commands called items. Many different menus, each controlling a specific set of tasks, can exist in the user-interface system.

Menus are special type of segments because the segment and its associated primitives can be animated graphically and the bounding boxes of each primitive are in display coordinates rather than world coordinates. This feature provide quick animation of graphical items without regaining the program to recalculate the display regions identified with the cursor. When the cursor enters the bounding box of menu item, the item is highlighted (See fig. e). When the cursor leaves the bounding box, the highlighting is dimmed and the menu animation system tests the other items to find the cursor, which designates a new item to animate. To select an item, we simply click when the cursor is on that item. By using menus for command selection, we can point at both graphical items and commands in a consistent manner.

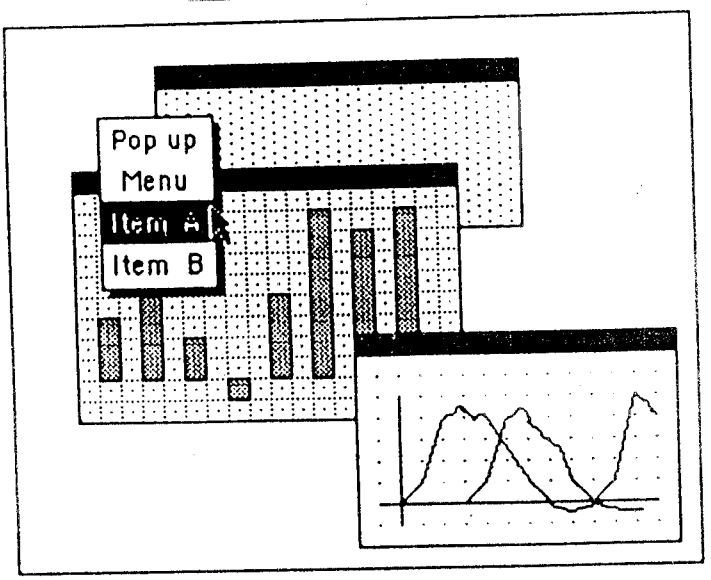


Fig g. A pop up menu

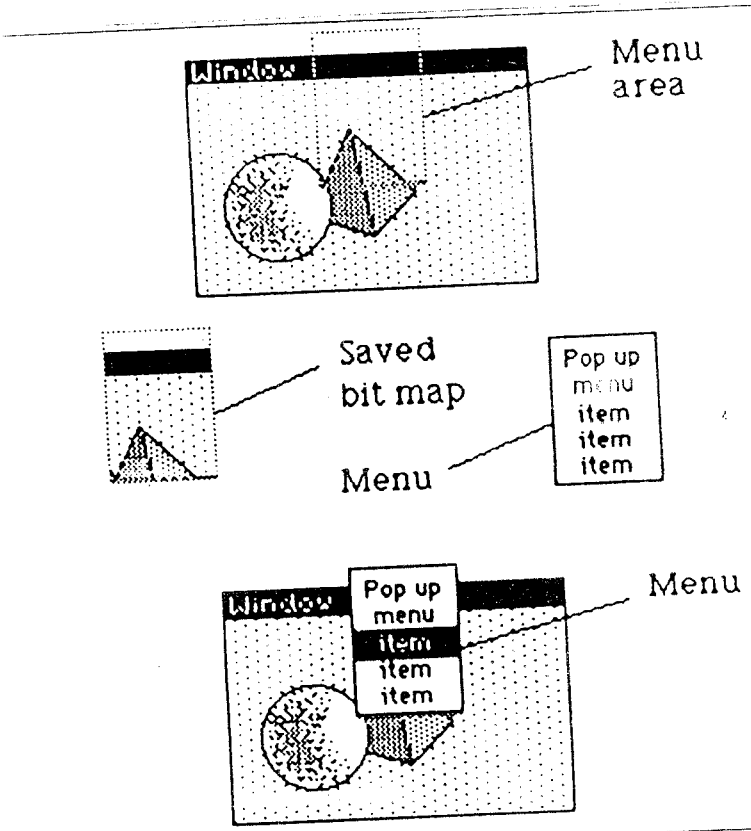


Fig h. Menu bit map elements

Many different methods of displaying and animating menus and their items are available. Usually, the type of menu an application uses depends on the application's purpose. If the application is an educational program, the commonly used menu items should be displayed at all items.

Menus can be grouped into three basic categories. Static, pop-up and pull-down. Static menus are the easiest to implement. The items of static menu are displayed on the computer screen at all times (See fig. f).

Pop-up menus get their name from the way they are activated (See fig. g). To activiate a pop-up menu, place the cursor in a special graphical region and click. The menu then 'pops-up' on to the screen under the cursor. The function pointers in the window structure often direct events to bigger pop-up menus that request further action from the user, when a menu appears, the program is ready for us to select an item. When an item has been selected, the menu disappears. To allow the menu to be erased and the display are that the menu obscures to be restored, the function must save this area of the bit map, another part of memory before the menu is drawn. Then, when the item seletion is completed, the previous bit map contents can be restored (See Fig h). This type of menu requires us to remember wich graphical regions invoke which menus.

The preceding two menu styles can be combined to form a hybrid called the pull-down menu. This menu incorporates the visual properties of the static menu and the efficient display of the pop-up menu. Each menu is given a name which serves as a menu title. For example, a menu used for editing may be named edit.

The titles of the menus are displayed in a single row across the top of the screen. When we place the cursor over a title and click, the corresponding menu pops up directly beneath the title (See Fig. i). If no item is selected before the cursor leaves menu's area, the menu disappears and the bit map is restored. By combining static menu names with dynamic menu items, we achieve a hierarchial method of command selection.

More than one type of menu can be used in one application program, and menu types can be commbined. For eg., a static menu can invoke pop-up menus. The static menu is the easiest to implement because no special bit map save and restore functions are needed. Pop-up and pull-down menus, however, offer greater flexibility. With the correct combinations of menus, a user interface can be a consistent and friendly tool.

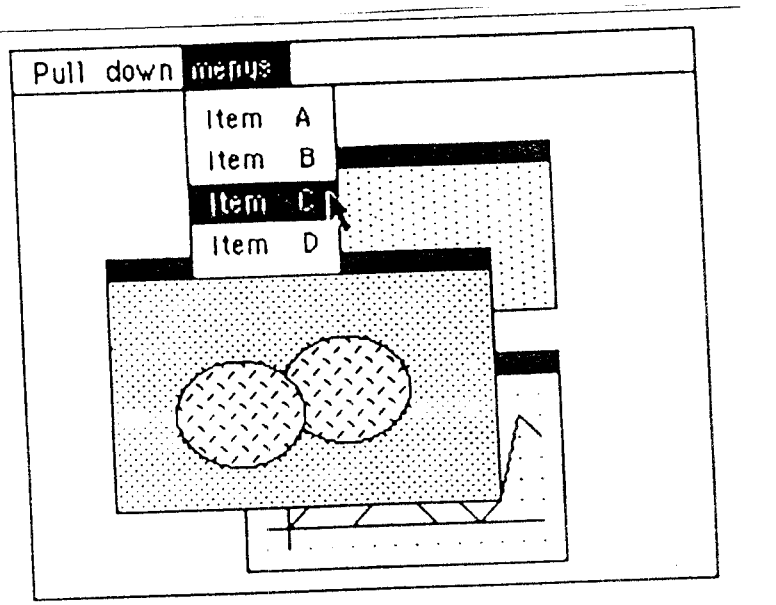


Fig 1. A pull down menu.

C. The Altos System Interface

Altos System V/386 continues Altos' technological leadership by offering a powerful UNIX/XENIX merged product that combines AT&T's UNIX standards, Microsoft's refinements for commercial use, and Altos' enhancements, which take advantage of the Altos system architecture and provide ease of installation, use, and maintenance of Altos systems.

Although the roots of AT&T's UNIX can be traced to the late '60s, the operating system first became commercially available in the late '70s. Microsoft's UNIX-derived operating system, XENIX, became the leader in the proliferation of UNIX and UNIX look-alike products to follow on the Intel architecture. Altos, a leader in recognizing the power of UNIX, has supplied the XENIX operating system for our Intel based (8086, 80286 and 80386) and the UNIX operating system on our Motorola-based computers (68000 and 68020) since 1982.

Software developers and end-users benefit with Altos System V/386 in several ways. Existing software that operates on Altos XENIX System V/386 will run without modification under Altos System V/386 eliminating the costs associated with developing compatible software. For software developers, increased standardization and portability maximizes the return on your development dollars by minimizing your maintenance and porting costs. End-users benefit from the increased standardization because a wider range of software will now be available.

Altos Enhancements

Included with Altos System V/386 is the Altos Office Manager (AOM) a powerful and easy-to-use menu system that eliminates the need for any direct interaction with the operating system. Altos computers can be configured so that AOM Menu System automatically appears when you log on to the system. The AOM Menu system is comprised of eight "pages" of screens, with four quadrants each, called "menus." Individual applications are easily installed in each menu, with up to six functions displayed as separate "point and pick" selections. Applications may be easily added, moved or removed from the menu system by the System Administrator through a special AOM page called the Menu Manager.

The AOM Menu System enhances Altos System V/386 in two important areas—system administration and program selection.

System Administration

With the AOM Menu System, the System Administrator can complete system maintenance tasks quickly and easily without prior UNIX knowledge:

Program Installation—All Altos applications are designed for simple installation under the control of the AOM Menu System.

System Configuration—Adding users or configuring ports is a quick and painless process using the AOM Menu System.

File Backup and Restore—Predefined backup routines provide backup of individual files and directories, the complete UNIX file system or files modified since the previous backup.

UNIX Utilities—Easy-to-understand AOM Menu selection such as "Create New Directory" or "Change File Permissions" replace advanced operating commands such as mkdir and chmod. The new or occasional user works effectively without mistakes or constant references to the manual.

Special Utilities—Special functions on the Menu Manager page, such as setting permissions on certain menus and changing the name of a page, give the System Administrator flexibility to configure and maintain the AOM Menu System.

Program Selection

The new or occasional user will find the AOM Menu System a friendly way to access often-used programs.

Applications are initiated by moving the cursor to the desired function with the arrow keys and pressing RETURN. Upon exiting the program, the AOM Menu System returns and awaits the next selection.

AOM Tool Kit

Available separately, the AOM Tool Kit provides the tools and instructions needed by the Altos reseller or software developer to perform the following functions:

Integration of Vertical Applications—Third-party developed software to install and operate from the AOM Menu System as easily as Altos-supplied software. The Altos reseller is able to provide a professional, turnkey solution.

Customization—Menu displays can be customized for installation by adding text or changing terminology, including translation to other languages.

COMPONENTS	FEATURES
CPU	68030 based 32 bit architecture (with floating point co-processor)
Main memory	4 MB
Virtual address space	Upto 256 MB per process
Colour display features	19 inch 1024 X 800 colour display
Peripheral support	170 MB winchester disc 60 MB cartridge tape One RS 232-C port Key board and Mouse