THE DEVELOPMENT AND ANALYSIS OF A PORTABLE RUN TIME LIBRARY ACCESSABLE TO ALL FORTRAN, COBOL AND PASCAL COMPILERS UNDER THE UNIX SYSTEM 5 OPERATING SYSTEM

by

KENDALL ROBERT PAYNE

B.S., Baker University, 1982

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1984

Approved by:

Major Professor

TABLE OF CONTENTS

THIS BOOK CONTAINS NUMEROUS PAGES WITH THE ORIGINAL PRINTING BEING SKEWED DIFFERENTLY FROM THE TOP OF THE PAGE TO THE BOTTOM.

THIS IS AS RECEIVED FROM THE CUSTOMER.

L I S T O F T A B L E S

C H A P T E R 1

INTRODUCTION

In this paper we are going to discuss the development of a portable run time library system that will support requirements of different compilers under the UNIX operating system. We are going to study in depth the run time libraries for COBOL, Fortran and Pascal. The purpose of doing this study is to determine the feasibility for providing a common collection of run time routines written in a low level language such as C or TAL(Transaction Application Language) that could be available in program environments on any target machine that runs under the UNIX system 5 operating system. We have included seven different compilers in our study of run time libraries. We analyzed the run time library routines in each of the seven libraries and developed a portable run time library that contains a total of 168 routines. The 168 routines is the number of unique routines in the library. The 168 routines contains all of the library routines from the seven compilers with some overlap among the routines. This can be seen by looking at Appendix A.

1.1 LANGUAGES USED IN THIS STUDY

As was mentioned in the previous paragraph, we chose Fortran, COBOL and Pascal as the languages we are going to analyze in our study. There are two principle reasons for choosing the three above mentioned languages. The first is that it gives us a wide cross-section of programming languages

that are used in a wide variety of applications. The second reason is that these three languages are the most widely used high-level languages in industry today.

1.1.1 FORTRAN

Fortran is a highly scientific language used primarily by engineers, architects and physicists for scientific applications. We will study two different Fortran compilers in this report, IBM and Tandem Fortran. The Fortran libraries contain primarily arithmetic routines and some conversion routines. The Tandem and IBM Fortran compilers give one a good representation of the types of routines found in Fortran run time libraries.

1.1.2 COBOL

COBOL, on the other hand, is primarily used for business applications. In developing a common library of routines that interface with three different languages, we have included COBOL in the group because COBOL routines provide a library with a great deal of diversity. The COBOL language is different from the other languages in some significant ways. It's data structure types and record formats are different, but more importantly, the COBOL run time library is radically different from the Fortran and Pascal run time libraries. The three principle types of routines found in any COBOL run time library are: conversion routines, internal arithmetic routines and COBOL verb routines. The routines found in the COBOL run time library are very distinguishable from those found in Fortran and Pascal libraries. All COBOL run time libraries contain essentially the same set of routines(we will discuss these routines later in the report). This is the reason why we have included only one COBOL compiler in our study.

1.1.3 PASCAL

Pascal is mainly used in our educational institutions and in private industry. It is good to use as a teaching tool because of it's strong typing and abundant number of data structures. Pascal is becoming quite popular in private industry today. It is being marketed on many home computers. In this report we have studied Pascal compilers that have come from four different machines. The four that were used were: Tandem, IBM, ACT and Perkin Elmer. Four different systems were used so that we could get a good cross-section of the routines that are found in the library of a Pascal compiler. Each Pascal compiler is unique in that it's routines are slightly different than the routines of any other Pascal compiler.

1.2 MACHINES USED IN STUDY

We are incorporating the use of four different machines in this study. The computers that were used are: Tandem, IBM, ACT and Perkin Elmer. These machines were chosen primarily because they are representative of the machines found out in industry today. The Tandem Nonstop II system was included because we did an implementation of the I/O run time routines on the Tandem system for a Pascal compiler being developed for the Department of the Navy. The Tandem Nonstop system is a relatively new machine and is becoming widely used in industry today. ACT is a computer used by DARCOM ALMSA Corporation which is a division of the Department of the Army. IBM is currently one of the largest corporations in the country. It has a well-established tradition in the computer industry. They continue to market newer and better products all of the time. IBM is also very diversified, producing software and hardware for a wide range of machines from home computers to large mainframes such as the new 4381 series. Perkin

Elmer equipment is also very widely used among large corporations. Perkin Elmer software is versatile in that it will run in many different environments.

TABLE 1

COMPILERS USED IN STUDY

LANGUAGE	MACHINE
Fortran	IBM
	Tandem
Pascal	IBM
	Tandem
	ACT
	Perkin Elmer
COBOL	IBM

In the following chapters we are going to look at: the different types of routines found in a portable run time library, the subroutine parameters of the run time routines, study the run time routines in the UNIX operating system, and finally study the security, system recovery, file integrity, and overall performance of the UNIX operating system.

C H A P T E R 2

TYPES OF SUBROUTINES IN RUN TIME LIBRARY

In this chapter we are going to look at the different categories of routines that are in a common run time library. There are 12 different types of routines in the library: input/output, file handling, storage management, conversion, arithmetic, logic, data transfer, control, COBOL Verb, Miscellaneous, the Pascal Prefix and the Error handling routine.

TABLE 2

TYPES OF ROUTINES IN COMMON LIBRARY

Input/Output	File Handling
Storage Management	Arithmetic
Logic	Control
COBOL Verb	Data Transfer
Conversion	Pascal Prefix
Miscellaneous	Error Handling

2.1 INPUT/OUTPUT ROUTINES

The first type of routines that we will look at is the input/output routines. I/O routines deal primarily with the reading and writing of data. There are 26 different I/O routines. Pascal is very strong in the area of I/O. The Pascal compilers studied in this paper (with the exception of IBM) have a great number of I/O routines in their libraries. This indicates that I/O plays an important role in the Pascal language. In addition to the standard "read" and "write" routines there are many other forms of read and write routines contained within the libraries (see Appen-

dix A). There is also two forms of GET's and PUT'S . Each of the four Pascal compilers has at least one GET and PUT routine in it's library. The Tandem compiler has both forms of GET's and PUT's which allows one to work with both random and sequential files. Neither Fortran compiler has any I/O routines. The IBM COBOL compiler has three I/O routines: the write routine, the read routine and an I/O routine used for BSAM files. If one is going to do a lot of I/O then a Pascal compiler would be best suited to meet those needs.

2.2 FILE HANDLING

Since file handling is related to input/output in many important ways; we will discuss file handling routines next. File handling routines, as the name implies, work with the manipulation of files. Text, stored in one form or another is very often inputted or outputted to some type of file. As was the case with the input/output routines, the Pascal library also supports many different file handling routines. All four Pascal compilers have the standard file routines EOF (end of file) , EOLN (end of line), RESET, which moves the file pointer to the beginning of the file for reading, and REWRITE, which reinitializes files for writing. ACT Pascal allows more file management capabilities than the other three Pascal compilers do. ACT contains the routines DONEFILE, GETPREASS, PUTPREASS and CHECKOPEN and TEXTSTART which all perform maintenance on a file system. Perkin Elmer Pascal has a couple of routines that initialize file control blocks for both internal and external files. The Fortran and COBOL compilers do not deal with file handling much at all. The IBM COBOL compiler contains only the open and close routines. The Tandem Fortran compiler has one routine in it's library that assigns a file number to a given file. As can be seen

from Appendix A, Pascal is the dominant language to use for both input/output and file handling.

2.3 STORAGE MANAGEMENT

Storage management is also another very important aspect of the Pascal library. Storage management routines are concerned with the creation and destruction of memory blocks and the management of internal memory space within the computer. Pascal as was the case with I/O and file handling routines, has a great facility for storage management routines. ACT Pascal contains nine storage management routines and IBM Pascal contains ten routines. The routines NEW, RELEASE, and MARK are contained in all four versions of Pascal. IBM Pascal, besides having the basic storage routines, concentrates on routines that deal with memory size. SIZEFREE, SIZEOF, SIZESTACK, and SIZEHEAP all deal with the memory of the machine. ACT Pascal has routines that deal mainly with pointers, creation of records and blocks of storage. Tandem and Perkin Elmer Pascal both have just the basic routines for storage management. The two versions of Fortran combined contain no storage management routines. IBM COBOL has only one routine RELEASE which deallocates space that was occupied by a given variable P.

2.4 CONVERSION ROUTINES

The next type of routines that we are going to look at are the conversion routines. The conversion routines are designed to convert a number from one form to another. An example of a conversion is a translation of a binary number to a decimal. Conversions are done in both internal and external form. Most of the conversion routines contained in the common library come from COBOL. The routines involve conversions on decimal numbers, binary numbers internal and external floating point numbers in COBOL and integer to reals and reals to integers in Fortran. COBOL has a great facility for conversion routines. As can be seen from Appendix A, the COBOL library contains 21 conversion routines. None of the four versions of Pascal contain any conversion routines except for ACT Pascal which has one routine for evaluating real numbers. The IBM version of Fortran has two conversion routines. The conversion of an integer to a real number and vice versa. Tandem Fortran has no conversion routines.

2.5 ARITHMETIC ROUTINES

The next type of routines that we are going to study is the arithmetic routines. The arithmetic routines are used to perform mathematical functions. Both versions of Fortran used in this study contain a great number of arithmetic routines. Tandem Fortran contains 26 different routines and IBM Fortran has 29 routines. This is many more than any of the other compilers have. Both libraries have: the common and natural logarithmic functions, the square and square root functions, sine, cosine, tangent, the arc functions, the hyperbolic functions, gamma functions, absolute value, minimum and maximum value, and many functions for complex numbers. These routines play an important role in many engineering applications and in the

hard sciences. The Tandem and IBM Pascal compilers have many of the more standard mathematical routines, SINE, COSINE, etc. They both also have some character manipulation routines, which neither Fortran compiler contains. Both the Tandem and IBM libraries contain the routines CHR, PRED, ORD, and SUCC. The ACT Pascal compiler contains only the essential arithmetic routines, SINE, COSINE, ARCTANGENT, Natural logarithm, exponential and square root. Perkin Elmer Pascal has no arithmetic routines. The IBM COBOL compiler has four arithmetic routines. All of them involve internal calculations using 30-bit binary numbers(see Appendix A). This allows one to do internal decimal division, multiplication and floating point exponentiation. These are things that the Fortran and Pascal libraries do not provide for.

2.6 LOGIC

in this section we shall look at the logic routines. Logic routines are routines that are used to perform logical operations between sets. Pascal is the only one of the three types of compilers in this study that contains any logic routines. There are five different logic routines that are associated with the Pascal compilers talked about in this paper. Perkin Elmer Pascal has routines for COMPARE, AND, OR, and DIFFERENCE. These are all routines that perform operations on sets. The "set" is an important data type in Pascal. IBM Pascal contains the routine CARD, a routine that determines the cardinality of a set. See Appendix B for a description of the logic routines.

2.7 DATA TRANSFER ROUTINES

The data transfer routines are routines that transfer data from unpacked arrays to packed arrays and vice versa. There are two routines in this group. There are separated into this class because they are unique and do not fit into any other category. The routines are "pack" and "unpack". They are data types in Pascal. Pack takes an array of characters and copies it into a packed array. Unpack does just the opposite; it copies data from a packed array to an unpacked array. Both routines appear in IBM Pascal.

2.8 CONTROL ROUTINES

These routines are put into the class "control routines" because they deal with printer overflow and spacing, hence the names printer overflow and printer spacing (see Appendix B). The printer overflow routine is used to control printer overflow testing and page ejection. The printer spacing routine, as the name implies, is used to control printer spacing. Both control routines are found in the IBM COBOL run time library.

2.9 PASCAL PREFIX

In this section, we will look at the Pascal Prefix. The Pascal prefix is unique to Pascal and is a set of procedure heading declarations and the necessary CONST and TYPE declarations. The Prefix is added at the beginning of each Pascal program and is used to call any of the supporting routines. All four versions of Pascal being studied contain the Pascal prefix.

2.10 COBOL VERB ROUTINES

Next we will look at a group of library routines known as the COBOL verb routines. These routines all have a common identity; when executed, they perform a function within the COBOL language. The verbs MOVE, ACCEPT, DISPLAY, PERFORM, and STOP RUN are some of the standard functions and appear in almost every COBOL program written. Other COBOL verbs such as TRANSFORM, CLASS TEST, and SEGMENTATION appear frequently in COBOL programs. With the exception of the SORT routine, these routines would be of very little use to one using Pascal or Fortran. The sort routine would be useful if one were to sort a file that contained thousands of records. The sort routine along with all of the other COBOL verb routines would be needed in order for one to have a common library in which COBOL can interface with.

2.11 MISCELLANEOUS ROUTINES

These routines are categorized as miscellaneous routines because they do not fit into any other of the previous ten classes of routines that have been discussed in the last several sections. One can look at these routines as "leftover" routines that are often used for special purposes. There is a wide variety of routines in this category. IBM Pascal is well known for it's general-purpose routines. IBM Pascal provides one with a lot of diversity as a result of these routines. IBM Pascal has 14 different miscellaneous routines. Tandem Pascal has one routine, namely CLOCK, which keeps track of time according to Greenwich mean time. Neither Fortran compiler has any miscellaneous routines. The IBM COBOL compiler has the routine DATE which is a call to the system to obtain the current data in YY/MM/DD format. As one can see from Appendix A, IBM Pascal provides the

most support for miscellaneous routines.

2.12 ERROR HANDLER

The last type of run time routines we are going to look at is the error handler. Each of the seven compilers in this study have an error handling routine. The function of the error handler is to provide a place to go in the event that a call is made to a routine that is not contained in the common library. When a call is made to a routine that is nonexistent, one enters the error handler. An error message is emitted and the process is either terminated or given the opportunity to make a call to a different library module depending on the seriousness of the error. The error handler is designed to handle exceptional cases. The error handling routine plays an important role in the common library because there are always exceptions in any programming environment.

In the last several paragraphs, we have discussed the 12 different types of routines found in a common run time library. Each type of routine plays an important role in providing a well-rounded run time library. In chapter three, we are going to look at the different types of subroutine parameters.

C H A P T E R 3

SUBROUTINE PARAMETERS

In the previous chapter, we looked at the different types of routines in a common run time library. In this chapter we are going to look at subroutine parameters. There are two different kinds of subroutines found in run time libraries, procedures and functions. We will first discuss functions and then four different categories of procedures.

3.1 FUNCTIONS

A function is a block of statements within a program which form an independent and separate component. This block of statements constitutes a subprogram which performs a particular set of operations on a supplied set of arguments and returns a single value. Each time the function is invoked, control transfers to the block of statements defined for that function. After these statements have been performed, execution returns to the statement from which the function was called. In the common library that we have designed, the routines that fall into the category of functions are the conversion, logic and arithmetic routines. The memory size routines within the storage management routines are also functions. Of the file-handling routines, there are only two functions: EOF and EOLN, which return the boolean values true or false. The return values of the memory size routines are of type integer. The return values and parameters of the conversion routines are of many different types: integer, real, floating point and binary. The parameters and return values for the arithmetic routines likewise consist of heterogeneous data types: integer, real, complex,

double-precision real, boolean and char. However most of the data types are the numeric data types: integer, real and complex. The data types of the logic routines can be composed of any definable type. As we have seen, functions appear in many different places within the run time library, but are most prominent in the area of conversion and arithmetic routines.

3.2 PROCEDURES

Procedures are subroutines that, when invoked, execute a block of statements and return a number of values to the calling program. These values are passed back to the calling routine through parameters. There may be one or more parameters in a procedure heading. Procedures, when invoked, transfer control to the statements of the procedure's definition and establish the correspondence between arguments and parameters. After execution of the subroutine statements, control is returned to the statement which immediately follows the statement which invoked the procedure. In this paragraph and the ones to follow, we will look at procedures within the common run time library. There are four different categories of procedures. We have classified them into groups of procedures with no parameters, one parameter, two parameters and multiple parameters.

3.2.1 PROCEDURES WITH NO PARAMETERS

The first type of procedure we are going to look at is those procedures that have no parameters. The procedures within the run time library that fall into this category are the initialization, utility and control routines. The initialization routines make sure all necessary files are open, the file tables are initialized and the control blocks are set up and initialized. The control routines that were discussed earlier

are used to control printer overflow testing and printer spacing. The third set of routines that fit into this category are the utility programs. These all come from the IBM COBOL library. They are the SORT, SEGMENTATION, CHECKPOINT and STOP RUN. As can be seen from Appendix B, none of these require any parameters. There are few run time routines that have no parameters. Most of the run time routines have one or more parameters. They are more complex and require that data be passed back and forth. These procedures will be discussed in the following paragraphs.

3.2.2 PROCEDURES WITH ONE PARAMETER

In this paragraph, we will look at procedures with just one parameter. This one parameter can be of many different types. Most of the one-parameter procedures are found in the I/O, file handling and miscellaneous routines. The I/O routines contained in this category are the GET and PUT routines and the READLN and WRITELN routines. All of the I/O routines declare the variable to be of type "filetype". "Filetype" corresponds to the type of the file that is to be created. Many file handling routines also fit in the one-parameter group. As was the case in the I/O routines, the parameter variables are of type "filetype". Most of these procedures came from the ACT Pascal compiler. A few of the storage management routines have procedures with only one parameter. The RELEASE, MARK and REMOVE routines contain just one parameter. The parameter types are normally of type text or of type item-type. The miscellaneous routines are the last large group of one-parameter routines. The main routines in the set of miscellaneous routines that have one parameter are the date and time routines. All three of these pass a string of characters as the parameter. The SEEK routine, which generates random numbers, also has just one parame-

ter. It passes a parameter of type integer. There are a few other system routines such as MESSAGE, HALT SETCC and TRAPATTN that contain just one parameter in their procedure declarations.

3.2.3 PROCEDURES WITH TWO PARAMETERS

In the last paragraph, we looked at procedures that have just one parameter. In this section we are going to look at procedures that have two parameters. The principle group of run time routines that fall into this category are the I/O routines that involve reading and writing bytes and characters and the storage management routines. The first parameter corresponds to a file which is to be read from or written to or a structure that is to be created. The second parameter gives us the location in which the data is read from, to be stored at, the type of data being read or written, or gives us the dimensions of the structure that is to be completed. For example, the read routines all have file-type as the first parameter(see Appendix A). The second parameter corresponds to the type of data being read. The write routines also have two parameters. Like the read routines, the first parameter is of type "filetype" and the second is of the type of data that is being written. The BSAM read used for COBOL has the first parameter again being defined of type "filetype" while the second parameter consists of an array of characters. Many of the procedures that involve creating new blocks and records have two parameters in their procedure declarations. The routines NEWBLOCK, NEWRECORD, STORESIZE, NEWINIT and SEEK all are two-parameter procedures. For a description of these routines, see Appendix B. The first parameter in NEWBLOCK and NEWRECORD is of the type of structure that is to be created; the second parameter is of type integer which corresponds to the number of bytes that

need to be allocated to the new structure. The routine STORESIZE has as it's parameters, the variable P which is of type pointer-type and second parameter size-to-store which is of type integer. "Size-to-store" corresponds to the number of bytes of data that need to be stored. NEWINIT and SEEK have as their first parameters "filetype". NEWINIT has the variable R, which is of type "rectype", as it's second parameter, while SEEK has seek-location which is of type integer as it's second parameter.

3.2.4 PROCEDURES WITH MULTIPLE PARAMETERS

The procedures that have more than two parameters, have been grouped into a category called procedures with multiple parameters. Most of the multiple parameter procedures occur in the input/output and storage management run time routines. The I/O routines of multiple parameters consists of read and write routines that work with strings and numerical data. In the case of each routine, the first parameter corresponds to the type of file the data is read from. The second, third and remaining parameters correspond to the variable that the data is to be read from or written to and the dimensions of the variable. The storage management routines that have multiple parameters are the Pascal NEW and DISPOSE routines. In each case, the first parameter corresponds to the pointer type. The second parameter corresponds to the size of the structure that is to be created or destroyed and the following parameters are the ordinal data types. A third set of procedures that have multiple parameters are the data transfer routines PACK AND UNPACK. In both cases, the first parameter corresponds to the array that is to be packed or unpacked. The second parameter corresponds to the size of the array that is to be packed or unpacked. The rest of the parameters are used to perform the other maintenance functions

of the packing/unpacking process.

In this chapter we have studied two different types of subroutines: procedures and functions. We divided procedures into four different groups: procedures with no parameters, one parameter, two parameters and multiple parameters. We have summarized this information on subroutines versus types of routines in table three below.

TABLE 3

SUBROUTINE PARAMETERS

SUBROUTINE	TYPES OF ROUTINES
FUNCTIONS	Arithmetic Logic Conversion Storage Management(memory size)
NO PARAMETER PROCEDURES	Initialization Utility Control Error Handler
ONE PARAMETER PROCEDURES	Input/Output (get,put,readln,writeln) File Handling Miscellaneous
TWO PARAMETER PROCEDURES	Input/Output(bytes & characters) Storage Management
MULTI PARAMETER PROCEDURES	Input/Output (strings & numerical data) Storage Management Data Transfer

C H A P T E R 4

UNIX SYSTEM 5 LIBRARY ROUTINES

In the last several chapters we have been discussing the routines and parameters of the common run time library. Now we are going to turn our attention to the routines that are in the UNIX system 5 run time library. The common run time library is going to interact with the UNIX operating system. Therefore, we need to study the UNIX operating system routines.

4.1 ROUTINES IN THE UNIX LIBRARY

The routines that are in Appendix C are routines that are common to the the UNIX system. These routines are functions that directly invoke UNIX system primitives. The library under the UNIX system is used to: maintain the file system, to create and destroy processes, keep a log on all processes currently running in the system, and to keep a schedule of all processes that are running or determine who can do what and when. The run time library also updates the directory system under UNIX which is a hierarchical structure that begins with a root directory.

4.2 UNIX LIBRARY VERSUS COMMON LIBRARY

There is little in common between the UNIX run time library and the common library of run time routines for COBOL, Fortran and Pascal. There are five routines that are common between the two libraries: READ, WRITE, OPEN, CLOSE, and TIME. The reason for the big difference in the libraries

is that the UNIX library is designed to maintain and keep track of all of the resources within the system and the common library developed for the above mentioned three languages is designed for use by individual users of the system. The portable run time library is used by individuals who are writing programs to run on the system.

The UNIX system could have some practical use for some of the routines in the portable run time library from time to time for use in maintaining the UNIX operating system. For example, UNIX could use the file handling routines to update files and directories. The storage management routines could be used when one creates and destroys blocks of memory. The control routines would be useful in controlling spacing and overflow by an I/O device. The miscellaneous routines in the common library also could be used for special purposes. For example, if one wanted to time the execution of a given subroutine within a program, he could insert the call for the routine CLOCK before and after the subroutine call and get the time it takes in milliseconds to execute the subroutines.

4.3 CAN COMMON LIBRARY BE PORTED OVER TO UNIX?

If one had a lot of time and patience, he could port the common routines(those routines not on UNIX) over to the UNIX library. The routines in the common library would have to be modified so that they could be integrated with the current routines under UNIX to form one package which runs on one machine. These modifications, however, would take a lot of time and patience since the routines in the common library come from several machines. In the common library that has been designed, there are seven different run time libraries representing four different machines. For each compiler we would have to port the software from the native

machine over to the host machine which in this case is UNIX. This would take a lot of time and dedication but it could be done. The routines that are directly common between the two libraries are in Table 4 below. For more detail on the Unix library routines see Appendix C.

TABLE 4

COMMON ROUTINES

(UNIX Library versus Portable Library)

Read
Write
Open
Close
Time

C H A P T E R 5

THE PORTABILITY OF THE UNIX OPERATING SYSTEM

In this chapter we are going to look at some of the features of the System 5 version of the UNIX operating system to determine whether it can operate in a portable environment or not. These features need to be checked out carefully in order to maintain the integrity of the UNIX operating system and to be sure the interfaces between UNIX and the common run time library work properly. The features we shall look at are: security, system recovery, file integrity and overall performance. We will look at each of these features in turn.

5.1 SECURITY

When the UNIX system was first developed security was not included with the package. The original system had a vast number of problems. Since that day efforts have been made to provide different means of security to the UNIX system.

5.1.1 PROTECTION AGAINST OVER-CONSUMPTION

The area of security in which UNIX is the weakest is in protection against system crash. The problem results in lack of checks for excessive consumption of resources. The most notable thing is that there is not a limit on the amount of disk storage used either in total space allocated, or in the number of files or directories allowed. In the system 5 version, users are prevented from creating more than a set number of processes simultaneously. Unless users are in collusion it is unlikely that anyone

can stop the system altogether. Excessive consumption of disk space files and swap space can easily occur accidentally in malfunctioning programs as well as at the command level. The UNIX system is defenseless against this kind of abuse. When the system does crash, there are methods that can be used to bring the system back up again. The methods of system recovery are discussed later on in this paper.

5.1.2 PROTECTION AGAINST UNAUTHORIZED PERUSAL

The protection of UNIX against use by unauthorized persons is much better than that against a sudden crash of the system. For protection, each UNIX file has associated with it eleven bits of protection information together with a user identification number and user group ID. Nine of the protection bits are used to determine an individual's rights for reading, writing and executing a given file. The last two bits of each file's protection information are called the set-UID and set-GID bits. The set-UID and set-GID bits are used so that one may write a program which is executable by others and which maintains files accessible to others only by that program. These are used only when the file is executed as a program.

There are a number of special cases involving access permissions. Since execution of a directory as a program is a meaningless operation, the execute-permission bit for directories is taken instead to mean permission to search a directory for a given file during the scanning of a pathname. If a directory has execute permission, but no read permission for a given user, he may access files with known names in the directory, but may not read(list) the entire contents of the directory. Write permission on a directory is interpreted to mean that the user may create and delete files in that directory. It is impossible for any user to write directly to any

directory.

The only major problem involving protection against unauthorized personnel involves that of the "super-user" who is able to read any file and write any non-directory. The superuser is able to change the protection mode and the owner UID and GID of any file to invoke privileged system calls.

5.1.3 MODES OF ACCESS

The first necessity for a secure system is arranging that all files and directories have the proper protection modes. Traditionally, UNIX software has been exceedingly permissive in this regard. Essentially all commands create files that are writable by everyone. In the current version, this policy may be easily adjusted to suit the needs of the installation or the individual user. Associated with each process and it's descendents is a mask which allows individual users to determine the mode of accessibility that they wish. The standard mask, set by login, allows all permissions to the user himself and to his group and no others.

5.1.4 PASSWORD SECURITY

On the issue of password security, UNIX is probably better than most systems. Passwords are stored in encrypted form which outside the serious attention by specialists in the field, provides a reasonable means of security. Since both the encryption algorithm and the encrypted passwords are available, exhaustive enumeration of passwords is feasible to a point. Users should choose a password that is at least six characters long and randomly chosen from an alphabet which includes special digits and special characters. The set-UID notion must be carefully used if any security is

to be maintained. The first thing to remember is that a writable-set-UID file can have another program copied onto it.

5.2 SYSTEM RECOVERY

In the System 5 version of UNIX, like any other system, crashes are bound to happen from time to time. When they occur, one needs to know the procedures for system recovery.

5.2.1 FILE SYSTEM

The first thing one does following a crash is to check the consistency of the file system. This is done by the File System Check Program(FSCK) which is an interactive file system check and repair program. FSCK is performed on all systems that were in use at the time of the crash. If any serious file problems are found, they should be repaired immediately.

If disks are in need of repair, the first thing that one should keep in mind is that an addled disk should be treated gently. It should not be mounted unless necessary. If the data on the disk is valuable, the disk should be copied before surgery is performed on it.

FSCK is adept at diagnosing and repairing disk problems. It first identifies all of the files that contain bad blocks or blocks that appear in more than one file. Any such files are then identified by name and FSCK requests permission to remove them from the system. FSCK will also report on incorrect link counts and will request permission to adjust any that are erroneous.

5.2.2 BRINGING SYSTEM UP

After the file system has been taken care of, one tries to boot the system up. The following steps are taken as a general strategy to get the system up and running again: boot an older system version and/or minimum configured system, boot from the backup root-file system, boot from another disk pack or the secondary disk drive and finally if all else fails, have the hardware checked out. If the system will not boot, it is caused by one of four things: hardware problems, an improperly configured system, a corrupted boot section on a disk, or a corrupted root file system.

5.2.3 MEMORY DUMPS

All messages printed by the UNIX system are saved in a circular buffer containing memory starting at the symbol PUTBUF. These messages can be looked at by examining the memory dump using crash(1m).

All file systems should be taken care of before attempting to look at the dumps. The dump should be read into the file with the pathname /usr/tmp/core. To print the process table at the time of the crash, one executes the command PS -EL -C /usr/tmp/core. One executes the command "WHO" to list the users who were on the system at the time of the crash.

5.3 FILE INTEGRITY

The integrity of the file system is maintained by continual updates to the file system and detection and correction of corruption within the system. Each of these will be be discussed in turn.

When a UNIX operating system is brought up, a consistency check of the file systems is always performed. This precautionary measure helps to

ensure a reliable environment for the file storage on the disk. The 5.0 file system features a large internal block size compared to a lot of systems. The block size is 1024 bytes instead of 512 bytes. This increases the performance of the I/O bound applications. The size of the internal system buffers is also 1024 bytes. For a 1024 byte block file system, data transfers to and from disk are in 1024 byte operations.

5.3.1 UPDATES OF THE FILE SYSTEM

Every working day hundreds of files are created, modified and removed. Every time a file is modified, the UNIX operating system performs a series of updates. These updates, when written on disk, yield a consistent file system. There are five types of system updates. They are: superblock, inodes, indirect blocks, data blocks(directories are files), and free-list blocks.

The superblock contains information about the size of the file system, the size of the inode list, part of the free-block list, the count of free blocks, the count of free inodes and part of the free-inode list. The superblock of a mounted file system is written to the file system whenever a file system is unmounted or a sync command is issued.

The inode contains information about the type of inode, the number of directory entries linked to the inode, the list of blocks claimed by the inode and the size of the inode. An inode is written to a file system upon closure of the file associated with the inode.

There are three types of indirect blocks, single-indirect, double-indirect and triple indirect blocks. A single-indirect block contains a list of some of the block numbers claimed by an inode. A double-indirect

block contains a list of single-indirect block numbers. A triple-indirect block contains a list of double-indirect block numbers. Indirect blocks are written to the file system whenever they have been modified and released by the operating system. More precisely, they are queued for eventual writing.

Data Blocks may contain file information on directory entries. Each directory consists of a file-name. Data blocks are written to the file system whenever they have been modified and released by the operating system.

The last form of update is the First-Free-List Block. The free-list blocks are a list of all blocks that are not allocated to the superblock, inodes, indirect blocks, or data blocks. Each free list block contains a count of the number of entries in this free-list block and a partial list of free blocks in the system.

5.3.2 DETECTION AND CORRECTION OF CORRUPTION

A file system can become corrupted in a variety of ways. The most common of these ways are improper shutdown procedures and hardware failures.

The elimination of corruption in the 5.0 file system is taken care by the File System Check Program(FSCK). FSCK uses the redundant structural information in the UNIX file system to perform several consistency checks. When the UNIX operating system is brought up, a consistency check of the file systems should always be performed. This precautionary measure helps to ensure a reliable environment for file storage on disk.

When a file system has been corrupted by one of the two ways mentioned above, the problem is detected and then taken care of. A quiescent file system(an unmounted system not being written on) may be checked for structural integrity by performing consistency checks on the redundant data intrinsic to the file system. The redundant data is either read from the file system or computed from other known values. A quiescent state is important during the checking of a file system because of the multipass nature of the FSCK program.

One of the most corrupted items is the superblock. The superblock is prone because every change to the file system's blocks modifies the superblock. The superblock and it's associated parts are most often corrupted when computer is halted and the command involving output to the system was not a sync command.

An individual inode is not as likely to be corrupted as the superblock. However, because of the great number of active inodes, there is almost as likely a chance for corruption in the inode list as in the superblock.

Indirect blocks are owned by the inode. Therefore, inconsistencies in indirect blocks directly affect the inode that owns it. Inconsistencies that can be checked are blocks already claimed by another inode and block numbers outside the range of the file system.

There are two types of data blocks, plain data blocks and directory data blocks. Plain data blocks contain the information stored in a file. Directory data blocks contain directory entries. FSCK does not attempt to check the validity of the contents of a plain data block. Each directory

data block can be checked for inconsistencies involving directory inode numbers pointing to unallocated inodes, directory inode numbers greater than the number of inodes in the file system, incorrect directory inode numbers for "." and ".." and directories which are disconnected from the file system. In addition, the validity of the contents of a directory's data block is checked.

Free list blocks are owned by the superblock, therefore inconsistencies in free-list blocks directly affect the superblock. Inconsistencies that can be checked are a list count out of range, block numbers out of range, and blocks already associated with the file system. If there is any problems with the free-block list, the FSCK may rebuild the list excluding all blocks in the list of allocated blocks that contain bad data.

Free list blocks are owned by the superblock, therefore inconsistencies in free-list blocks directly affect the superblock. Inconsistencies that can be checked are a list count out of range, block numbers out of range, and blocks already associated with the file system. If there is any problems with the free-block list, the FSCK may rebuild the list excluding all blocks in the list of allocated blocks that contain bad data.

5.4 OVERALL PERFORMANCE

The overall performance of the UNIX System 5 is comparable with that of other systems. In the area of security, UNIX does not provide very good protection against system crash, but provides adequate protection against unauthorized perusal and destruction of the system with exception of the superuser. Each process within the system and descendants of that process have what is called a "mask" which allows an individual to determine his

own mode of access. The password security provided by UNIX system 5 is better than that of most other systems.

UNIX does provide for system recovery if for some reason the system should go down. FSCK(M) is an function on the System-5 version of UNIX that detects and repairs problems with files. When all files have been repaired one then attempts to boot the system up. There is a set strategy that is used to boot the system following a crash. First one would boot an older system or a minimum configured system. If this does not work then one boots using the backup root system. If this also fails, one then tries to boot the system using another disk pack or a secondary disk drive. If all else fails, one then must check out the hardware.

The integrity of the file is maintained by two major components: file updates, and detection and correction of corruption within the system. There are five kinds of updates: superblock, inodes, indirect blocks, data blocks, and free-list blocks. A file system can be corrupted in two major ways: shutdown procedures and hardware failures. It is the job of the FSCK(File System Check Program) to detect and correct corruption within the file system. FSCK is able to eliminate most of all errors that occur within the file system. This maintains a high rate of integrity within the system.

C H A P T E R 6

CONCLUSION

6.1 REVIEW

In this paper we have studied the development of a portable run time library system that supports requirements of three different compilers running under the UNIX operating system. We first studied the twelve different types of run time routines that must exist in a common library in order for it to be functionable. The twelve different types of library routines are: input/output, data transfer, file handling, control, storage management, single used(COBOL), conversion routines, the prefix call, arithmetic, logic, error handling, and miscellaneous routines. As we have learned from earlier chapters, there is a wide variation among the types of routines in the library both in terms of the languages they support and the functions that they perform.

Next we studied the parameters of the run time procedures. In our study we divided them into five categories. Procedures with no parameters, procedures with one parameter, procedures with two parameters, procedures with multiple parameters and functions. The routines that fit the category of no-parameters routines are the initialization, COBOL verb and the control routines. The routines that are in the one-parameter group are the file handling and miscellaneous routines. Most of the I/O and storage management routines have more than one parameter. The arithmetic and conversion routines fit the category of functions.

In chapter four we looked at the routines found in the UNIX operating system library. We found that the routines here have very little in common with the routines in our portable run time library. The UNIX run time library is used to maintain the resources of the system. The portable run time library is designed to accommodate the needs of the individual user to the accomplish the tasks that he needs to have done.

In the last chapter, we studied the security, system recovery, file integrity and overall performance of the UNIX system 5 operating system. It is important to have a system that is secure and one that can be maintained on a daily basis. That ensures that it runs at maximum efficiency.

The primary function of the common run time library is to provide one with a great amount of flexibility in allowing individual users to write software in a portable environment using Pascal, Fortran and/or COBOL. A second function of the common library is to allow one to interface with a wide variety of run time routines. In all, there are 168 different routines that are needed in order to make the library portable. Of the three languages that are studied, COBOL has the most unusual set of routines in the conversion and COBOL verb routines. Including COBOL compilers in our library provides a great deal of diversity to the library.

6.2 FINAL REMARKS

In this paper we have discussed in detail the requirements needed for the development of a common run time library to be built under the UNIX operating system. It is possible to develop a run time library for all Fortran, COBOL and Pascal compilers provided that all of the above mentioned qualification are met. It would take a great deal of time and research to actually implement this portable run time library but it could be done if one is determined to do it.

A portable run time library that includes routines from three languages: COBOL, Fortran, and Pascal can be an important tool when on has to work in many different environments.

A P P E N D I C E S

APPENDIX A

SUBROUTINES FOR COMMON LIBRARY

Many of the routines appear in more than one compiler. What we have done in the tables listed in Appendix A is show the relationship between the run time routines versus the compilers. If a given compiler contains a certain routine then an 'X' is placed in that slot of the table. The 168 run time routines is the minimum number of routines that is needed for the common run time library to be portable and efficient.

LIST OF ABBREVIATIONS

TAN	TANDEM
PE	PERKIN ELMER
CBL	COBOL
Ext	External
Int	Internal
Bin	Binary
Float	Floating Point Number
pwr	power

INPUT/OUTPUT	PASCAL				FORTRAN		CBL
	TAN	PE	ACT	IBM	TAN	IBM	IBM
Read		X		X			
Write		X		X			X
Get	X	X	X	X			
Get'	X						
Put	X	X	X	X			
Put'	X						
Readcharacter	X	X	X	X			
Readbyte		X					
Readstring	X	X					
Readshortinteger	X	X					
Readinteger	X	X	X				
Readshortreal	X	X					
Readreal	X	X	X				
Readln	X	X	X				
Writecharacter	X	X	X				
Writebyte		X					
Writestring	X	X	X				
Writeshortinteger	X	X					
Writeinteger	X	X	X				
Writeshortreal	X	X					
Writereal	X	X	X				
Writeln	X	X	X				
Ioinit	X		X				
Iodata			X				
BSAM I/O							X
BSAM Read							X

FILE HANDLING	PASCAL				FORTRAN		CBL
	TAN	PE	ACT	IBM	TAN	IBM	IBM
Filenumber					X		
Filcpy		X					
IFCB (1)		X					
EFCB (2)		X					
Reset	X	X	X	X			
Rewrite	X	X	X	X			
EOF	X	X	X	X			
EOLN	X	X	X	X			
Extend	X						
Stcpy		X					
Open			X				
Close			X	X			X
Donefile			X				
Closeall			X				
Getpreass			X				
Putpreass			X				
Checkopen			X				
Program Entry	X						
Program Exit		X					
Textstart			X				

(1) -- File Control Block Initialization for internal files.
(2) -- File Control Block Initialization for external files.

STORAGE MANAGEMENT	PASCAL				FORTRAN		CBL
	TAN	PE	ACT	IBM	TAN	IBM	IBM
New	X	X	X	X			
New'				X			
Dispose			X	X			
Dispose'				X			
Release	X	X	X	X			X
Mark	X	X	X	X			
Newblock			X				
Newrecord			X				
Storesize			X				
Newinit			X				
Seek			X				
Remove		X					
Sizefree				X			
Sizeof				X			
Sizestack				X			
Sizeheap				X			

CONVERSION ROUTINES	PASCAL				FORTRAN		CBL
	TAN	PE	ACT	IBM	TAN	IBM	IBM
Ext. Float/ to Int. Decimal							X
Ext. Float/pt to Binary							X
Ext. Float/pt to Int. Float/pt							X
Bin. to Int Decimal							X
Bin. to Ext Decimal							X
Binary to Int. Float/pt							X
Binary to Ext. Float/pt							X
Int. Decimal to Ext. Float/pt							X
Int. Float/pt to Ext. Float/pt							X
Int. Decimal to Ext. Float/pt							X
Int. Decimal to Bin							X
Ext. Decimal to Bin							X
Int. Decimal to Int. Float/pt							X
Ext. Decimal to Int. Float/pt							X
Int. Float/pt to Int. Decimal							X
Int. Float/pt to Ext. Decimal							X
Int. Float/pt to Bin Int/pwr 10							X
Int. Float/pt to Binary							X
Int. Decimal to Sterling report							X
Int Decimal to Sterling non report							X
Sterling non report to Int Decimal							X
Integer to real						X	
Real to integer						X	
Evalreal			X				

ARITHMETIC ROUTINES	PASCAL				FORTRAN		CBL
	TAN	PE	ACT	IBM	TAN	IBM	IBM
Natural Logarithm	X		X	X	X	X	
Common Logarithm					X	X	
Exponential			X	X	X	X	
Square Root	X		X	X	X	X	
Arcsine					X	X	
Arccosine					X	X	
Arctangent	X		X	X	X	X	
Sine	X		X	X	X	X	
Cosine	X		X	X	X	X	
Tangent					X	X	
Cotangent						X	
Hyperbolic Sine					X	X	
Hyperbolic Cosine					X	X	
Hyperbolic Tangent					X	X	
Absolute Value	X			X	X	X	
Maximum Value					X	X	
Minimum Value					X	X	
Truncation	X			X	X	X	
Gamma						X	
Log Gamma						X	
Modulo					X	X	
Transfer of Sign					X	X	
Positive Difference					X	X	
Chr	X			X			
Linelength				X			
Nearest Whole No.					X		
D.P.Product					X		
Length	X			X	X		
Round	X			X	X		
Square	X			X			
Odd	X			X			
Ob. Real of Complex						X	
Ob. Imag of Complex					X	X	
Ob. Conj of Complex					X	X	
Exp real as comp.						X	
Precision Increase						X	
Pred	X			X			
Ord	X			X			
Succ	X			X			
Most Sig Real Root						X	
IDM(1)							X
IDD(2)							X
EIDBBE(3)							X
Floating pt. Exp							X

(1) -- Internal Decimal Multiplication
(2) -- Internal Decimal Division
(3) -- Exponentiation of Internal Decimal Base by Binary Exponent

LOGIC ROUTINES	PASCAL				FORTRAN		CBL
	TAN	PE	ACT	IBM	TAN	IBM	IBM
Compare		X					
And		X					
Or		X					
Difference		X					
Card				X			
DATA TRANSFER ROUTINES							
Pack				X			
Unpack				X			
CONTROL ROUTINES							
Printer Overflow							X
Printer Spacing							X
COBOL VERB ROUTINES							
Move							X
Transform							X
Class Test							X
Segmentation							X
Stop Run							X
Sort							X
Accept							X
Checkpoint							X
Display							X
Perform							X
PREFIX CALL	X	X	X	X			

MISCELLANEOUS	PASCAL				FORTRAN		CBL
	TAN	PE	ACT	IBM	TAN	IBM	IBM
Date			X	X			X
Juliandate			X				
Time			X	X			
Page	X	X		X			
Clock		X		X			
Wallclock				X			
Clockleft				X			
Seed				X			
Terminal		X					
Getparm				X			
Message				X			
Halt				X			
Setcc				X			
Trapattn				X			
System				X			
Undefined		X		X			
ERROR HANDLING	X	X	X	X	X	X	X

APPENDIX B

A SHORT DESCRIPTION OF LIBRARY ROUTINES CONTAINED IN THE COMMON RUN TIME LIBRARY

INPUT/OUTPUT ROUTINES

GET(F : file-type);
-- advances the current file position to the next component of a sequential disk file and assigns the value of the component to the file buffer.

GET'(F : file-type);
-- advances the current file position to the next component of a random disk file and assigns the value of the component the file buffer.

PUT(F : file-type);
-- appends the value of the buffer variable to a sequential disk file.

PUT'(F : file-type);
-- appends the value of the buffer variable to a random disk file.

READ(F : file-type, p1..pn : any-type);
-- transfers a data item or a set of data items from a terminal or file and assigns the data item(s) to a variable. The F represents a text-file and the p1..pn represent read-parameters.

READCHARACTER(F : file-type, c : char);
-- assigns a variable(of type char) the next character read from either a file or terminal.

READSTRING(F : file-type, str : string, str-size : integer);
-- assigns the variable STR(of type string) one or more characters read from either a file or terminal.

READBYTE(F : file-type, b : char<0:7>);
-- assigns a given variable the next byte read from either a terminal or file.

READSHORTINTEGER(F : file-type, s-int-value : integer);
-- converts a character string to an integer in internal form where the integer is in the range of -2**16 to 2**16.

READINTEGER(F : file-type, int-value : integer(32));
-- the same as readshortinteger except that the range is -2**32 to 2**32.

READSHORTREAL(F : file-type, s-real-value : real);
-- converts a character string to a real number stored in internal form. The range is +/-1.16*10**77 and the precision is 6 digits.

```
READREAL(F : file-type, real-value : real);
    -- the same as readshortreal except the precision
    is 11 digits.

READLN(F : file-type);
    -- assigns the file buffer the next character read after an
    end of line character.

WRITE(F : file-type, p1..pn : any-type);
    -- transfers a data item or a set of data items stored in a
    variable(s) to either a disk file or terminal.
    The F represents a text-file and the p1..pn represent
    write-parameters.

WRITECHARACTER(F : file-type, c : char);
    -- writes one character to a disk file or terminal.

WRITEBYTE(F : file-type, c : char<0:7>);
    -- writes one byte to either a disk file or a terminal.

WRITESTRING(F : file-type, str : string,
            str-size : integer, field-width : integer);
    -- writes a string of characters out to a disk file or
    terminal.

WRITESHORTINTEGER(F : file-type, ivalue : integer,
                  field-width : integer);
    -- converts the integer stored in internal form to a
    character string and writes it to either a file or terminal.

WRITEINTEGER(F : file-type, ivalue : integer(32),
             field-width : integer);
    -- same as writeshortinteger except that the precision
    is 11 digits.

WRITESHORTREAL(F : file-type, rvalue : real,
               field-width : integer, decimal-places : integer);
    -- converts an extended floating-point number to a
    character string and outputs the value to either a file or
    terminal.  The integer is displayed in exponential form.
    The precision is 7 decimal places.

WRITEREAL(F : file-type, rvalue : real, field-width : integer,
          decimal-places : integer);
    -- same as writeshortreal except that the precision is 16
    decimal places.

WRITELN(F : file-type);
    -- writes an end-of-line character out to a disk file
    or sends a carriage return to a terminal.
```

IOINIT -- this procedure reads the assign messages and sets up the file table. If any external file names are assigned to any of the logical file names in the file table, the external file is converted to internal form and placed in the appropriate entry in the file table. This procedure is called once by all object programs at the start of execution.

BSAM I/O(F : file-type);
-- (used strictly for COBOL programs) processes input/output statements for direct or relative files accessed sequentially.

BSAM READ(F : file-type, text : array[0..n] of char);
-- (COBOL) reads routine segments of a logical record and assembles them into a complete logical record.

FILE HANDLING ROUTINES.
==========

FILENUMBER(filenum : integer);
-- obtains a the file number from the system and assigns it to the given file.

FILCPY(F : file-type);
-- creates a backup or duplicate copy for a given file.

IFCB -- File Control Block initializing procedure for internal files.

EFCB -- File Control Block initializing procedure for external files.

RESET(F : file-type);
-- resets the current position to the beginning of the file.

REWRITE(F : file-type);
-- discards the current file-type and defines F to be an empty file.

EOF(F : file-type) : boolean;
-- returns a boolean value of end-of-file to the calling program. If end-of-file is reached the value TRUE is returned.

EOLN(F : file-type) : boolean;
-- returns a boolean value of end-of-line to the calling program. If end-of-line is reached the value TRUE is returned.

EXTEND(F : file-type);
-- advances the position of the file F to the end of the file so that additional components can be appended to the file.

STCPY(F : file-type);
-- copies a structure from one file to another.

OPEN(F : file-type);
-- opens a file.

CLOSE(F : file-type);
-- closes a file.

DONEFILE(F : file-type);
-- this procedure is called when the file F is no longer going to be needed.

```
CLOSEALL(F1..FN : file-type);
    -- simultaneously closes all files that are currently open.

GETPREASS(F : file-type);
    -- (PASCAL) checks that the file F satisfies the
    preassumptions for a get by the Pascal Standard.

PUTPREASS(F : file-type);
    -- (PASCAL) checks that the file F satisfies the
    preassumptions for a put by the Pascal Standard.

CHECKOPEN(F : file-type);
    -- checks to see that the file F satisfies the open before
    any I/O is attempted.

PROGRAM ENTRY -- (PASCAL) this procedure initializes the file
    tables for the files used in a given program and makes sure
    that all of the necessary files are open so that the program
    can begin execution.

PROGRAM EXIT -- (PASCAL) this procedure makes sure all files
    used in the program are closed and then terminates execution
    of the program.

TEXTSTART(F : file-type);
    -- this procedure tells the system what file is to be used
    until further notice is given.  This avoids the need to
    specify the file on every call.
```

STORAGE MANAGEMENT ROUTINES

```
NEW(P : pointer-type, size : integer,
    const c1..cn : ordinal-types);
    -- allocates a new undefined variable for the variable P
    which is of the domain-type P and assigns a reference to this
    variable for P.

NEW'(P : record of pointer-type, size : integer,
    const c1..cn : ordinal-types);
    -- same as NEW except that the domain type P must be a record
    type with variants.

DISPOSE(P : pointer-type, size : integer,
        const c1..cn : ordinal-types);
    -- indicates that the storage occupied by the variable P
    is no longer needed.  The variable P becomes inaccessible
    and P and all other pointers which reference that variable
    become undefined.

DISPOSE'(P : record of pointer-type, size : integer,
         const c1..cn : ordinal-types);
    -- same as DISPOSE except this procedure must be used in
    association with NEW'.

RELEASE(P : pointer-type);
    -- this procedure deallocates space that was occupied by the
    variable P.

MARK(P : pointer-type);
    -- stores in the pointer P a value that may be later used as
    the parameter of the procedure RELEASE.

NEWBLOCK(block : bufftype , blocksize : integer);
    -- creates a block of storage space of size "blocksize" to a
    given program.

NEWRECORD(record : rectype , recsize : integer);
    -- creates a new record for a given record type Rectype.

STORESIZE(P : pointer-type, size-to-store : integer);
    -- determines how much storage space is needed for a
    variable P which is of domain pointer-type.

NEWINIT(F : file-type, R : rectype);
    -- creates and initializes a new record R for the record
    type Rectype.
```

```
SEEK(F : file-type, seeklocation : integer);
    -- moves the current pointer to the location given in the
    parameters.

REMOVE(S : string);
    -- causes trailing blanks of a string S to be removed.

SIZEFREE : integer;
    -- returns the size in bytes of the unused area of storage
    between the stack and the heap.

SIZEOF(X : var/type identifier) : integer;
    -- returns the number of bytes of storage occupied by a
    variable of the same type as X.  X may be either a type or
    variable identifier.

SIZESTACK : integer;
    -- returns the size in bytes of the area of storage currently
    occupied by the stack holding local variables of active
    procedures and functions.

SIZEHEAP : integer;
    -- returns the size in bytes of the area currently occupied
    by the heap from which variables are allocated by the
    standard procedure NEW.
```

CONVERSION ROUTINES
==========

```
EXT. FLOAT TO INT. DECIMAL(X : float) : real;
    -- converts an external floating point
    number to an internal decimal.

EXT. FLOAT TO BINARY(X : float) : binary;
    -- converts an external floating point number
    to a binary number.

EXT. FLOAT TO INT. FLOAT(X : float) : float;
    -- converts an external floating point number
    to an internal floating point number.

BINARY TO INT. DECIMAL(x : binary) : real;
    -- converts a binary number to an internal
    decimal.

BINARY TO EXT. DECIMAL(X : binary) : real;
    -- converts a binary number to an external decimal.

BINARY TO INT. FLOAT(X : binary) : float;
    -- converts a binary to an internal floating point
    number.

BINARY TO EXT. FLOAT(x : float) : binary;
    -- converts a binary to an external floating point
    number.

INT. FLOAT TO EXT. FLOAT(x : float) : float;
    -- converts an internal floating point number
    to an external floating point number.

INT. DECIMAL TO EXT. FLOAT(X : real) : float;
    -- converts an internal decimal to an external
    floating point number.

INT. DECIMAL TO BINARY(X : real) : binary;
    -- converts an internal decimal to a binary number.

EXT. DECIMAL TO BINARY(X : real) : binary;
    -- converts an external decimal to a binary number.

INT. DECIMAL TO INT. FLOAT(X : real) : float;
    -- converts an internal decimal to an
    internal floating point number.

EXT. DECIMAL TO INT. FLOAT(X : real) : float;
    -- converts an external decimal to an
    internal floating point number.
```

```
INT. FLOAT TO INT. DECIMAL(X : float) : real;
    -- converts an internal floating point number to an
    internal decimal.

INT. FLOAT TO EXT. DECIMAL(X : float) : real;
    -- converts an internal floating point number
    to an external decimal.
INT. FLOAT TO BIN. INT/PWR 10(X : float) : binary;
    -- converts an internal floating point
    number to a binary integer and a power of 10 exponent.

INT. FLOAT TO BINARY(X : float) : binary;
    -- converts an internal floating point number to
    a binary number.

INT. DECIMAL TO STERLING REPORT(X : real) : sterling;
    -- converts an internal decimal to
    a Sterling report.

INT. DECIMAL TO STERLING NON-REPORT(X : real) : sterling;
    -- converts an internal decimal to
     a Sterling non-report.

STERLING NON-REPORT TO INTERNAL DECIMAL(X : Sterling) : real;
    -- converts a Sterling non-report
     to an internal decimal.

INTEGER TO REAL(X : integer) : real;
    -- (FORTRAN) converts an integer value to a real value.

REAL TO INTEGER(X : real) : integer;
    -- (FORTRAN) converts a real value to an integer value.

EVALREAL(X : real) : target-machine-real;
    -- converts the real number represented by character strings
     to the target machine representation of a real number.
```

ARITHMETIC ROUTINES
==========

```
    NATURAL LOGARITHM(x : integer/real/complex) : real/complex;
        -- returns the natural logarithm of x where x is of type
        real.  (y = ln x)

    COMMON LOGARITHM(x : integer/real/complex) : real;
        -- returns the common logarithm of x where x is of type real.
             (y = log(10) x)

    EXPONENTIAL(x : integer/real/complex) : real/complex;
        -- returns the value of the base of the natural logarithm
        raised to the power x. (y = e**x).

    SQUARE ROOT(x : integer/real/complex) : real/complex;
        -- returns the square root of x  (y = x**(1/2))

    SQUARE(x : integer/real) : real;
        -- returns the square of x  (y = x**2)

    ARCSINE(x : real) : real; -- returns the arcsine of x
           (y = arcsin x)

    ARCCOSINE(x : real) : real; -- returns the arccosine of x
            (y = arcos x)

    ARCTANGENT(x : integer/real) : real;
        -- returns the arctangent of x  (y = arctangent x)

    SINE(x : integer/real/complex) : real/complex;
        -- returns the sine of x  (y = sin x)

    COS (x : integer/real/complex) : real/complex;
        -- returns the cosine of x  (y = cos x)

    TANGENT(x : real) : real;
        -- returns the tangent of x  (y = tan x)

    COTANGENT(x : real) : real;
        -- returns the cotangent of x  (y = cot x)


    HYPERBOLIC SINE(x : real) : real;
        -- returns the hyperbolic sine of x  (y = sinh x)

    HYPERBOLIC COSINE(x : real) : real;
        -- returns the hyperbolic cosine of x  (y = cosh x)

    HYPERBOLIC TANGENT(x : real) : real;
        -- returns the hyperbolic tangent of x  (y = tanh x)

    ABSOLUTE VALUE(x : integer/real/complex) : integer/real/complex;
        -- returns the absolute value of x  (y = |x|)
```

```
MAXIMUM VALUE(x : integer/real) : integer/real;
    -- returns the maximum value of the set
    (y = max(x , x , x ..... x )) where x is either an integer or
    real.

MINIMUM VALUE(x : integer/real) : integer/real;
    -- returns the minimum value of the set
    (y = min(x , x , x ..... x )) where x is either an integer or
    real.

TRUNCATION(x : real) : integer/real;
    -- returns the value of x truncated to an integer value.
    (x must be of type real)
GAMMA(x : real) : real; -- returns the antiderivative of x

LOG GAMMA(x : real) : real;
    -- returns the log of the antiderivative of x.

MODULO(x,y : integer/real) : integer/real;
    -- returns the modulo of z  (z = remainder(x/y))

TRANSFER OF SIGN(x : integer/real) : integer/real;
    -- returns the resultant number * (-1).

POSITIVE DIFFERENCE(x : integer/real) : integer/real;
    -- returns the positive difference of x and min(x,y)
         (z = x  - min(x,y))

D. P. PRODUCT(x,y : real) : double(*);
    -- returns the double precision product of two numbers x and
         y.

   (*) -- A Double Precision real number.

LENGTH(c : char) : integer;
    --  returns the length of a character item.

LINELENGTH(f : text) : integer;
    -- returns the number of character positions in the remainder
    of the current line of f.

ROUND(x : real) : integer; --  returns to nearest integer to x.

NEAREST WHOLE NUMBER(x : real) : integer;
    -- returns the nearest whole number to x. X must
    be a positive real number.

ODD(x : integer) : boolean; -- returns true if the integer is odd

OBTAIN REAL OF COMPLEX(x : complex) : real;
    -- returns the real part of a complex number i. e.
    the x value in (z = x + yi).
```

```
OBTAIN IMAGINARY PART OF COMPLEX ARGUMENT(x : complex) : real;
    -- returns the imaginary part
    of a complex number  i.e. the y value in (z = x + yi).

OBTAIN CONJUGATE OF COMPLEX ARGUMENT(x : complex) : complex;
    -- return the conjugate of x + yi  i. e. x - yi.

EXPRESS REAL IN COMPLEX FORM(x : real) : real;
    -- express real number in the form x + yi
    where x is the real part and y is the imaginary part.  In
    this subroutine y = 0.

PRECISION INCREASE(x : real) : real;
    -- increase the number of digits a number is allowed
    to occupy.

PRED(x : ordinal-type) : ordinal-type;
    -- returns the predecessor of x.

ORD (x : ordinal-type/pointer-type) : integer;
    -- returns the integer ordinal number associated with
    the value of x.

SUCC(x : ordinal-type) : ordinal-type;
    -- returns the successor of x.

CHR(x : integer) : char;
    -- returns the character value whose ordinal value is x.

MOST SIG. REAL ARG(x : real) : real;
    -- obtain the most significant part of the real argument.

INT. DECIMAL MULTIPLICATION(x : real) : real;
    -- (COBOL) Internal Decimal Multiplication, multiplying
    a 30 digit number by a 30 digit number to obtain a product
    that is 60 digits in length and is stored internally in
    the system.

INT. DECIMAL DIVISION (x : real) : real;
    -- (COBOL) Internal Decimal Division, divide a 60-digit
    number by a 30-digit number to obtain the quotient which is
    stored internally by the system.

FLOATING POINT EXPONENTIATION (x : real) : real;
    -- (COBOL) returns a floating point number
    that is written in exponential form.

EXP INT DEC BASE BY BIN EXP(x : real) : real;
    -- (COBOL) Exponentiation of an Internal Decimal Base
     by a binary exponent.
```

LOGIC ROUTINES
==============

```
COMPARE(s1,s2 : set of any-type);
    -- performs a set compare on two different sets.

AND(s1,s2 : set of any-type) : set of any-type;
    -- returns a resultant set which is the intersection of
    two sets.

OR(s1,s2 : set of any-type) : set of any-type;
    -- returns a resultant set which is the union of two sets.

DIFFERENCE(s1,s2 : set of any-type);
    -- performs a set difference on two sets.

CARD(s : set-type) : integer;
    -- returns the number of elements in the set x.
```

DATA TRANSFER ROUTINES
======================

```
PACK(A : array[m .. n] of T, I : integer,
     Z : packed array [U..V] of T);
    -- copies data from an unpacked array to a packed array.
       The variables M, N, U, V are of type integer and the
       variable T is of any definable type.

UNPACKED(Z : packed array [U..V] of T,
         A : array [m..n] of T, I : integer);
    -- copies data from a packed array to an unpacked array.
       The variables M, N, U, V are of type integer and the
       variable T is of any definable type.
```

CONTROL ROUTINES
================

PRINTER OVERFLOW -- is used to control printer overflow testing and page ejection.

PRINTER SPACING -- is used to control printer spacing.

COBOL VERB ROUTINES

MOVE(T : text);
--(COBOL) used when one or both operands are variable in length. It is also used for READ and WRITE statements processed in conjunction with the same record-area clause.

TRANSFORM(Item : item-type);
--(COBOL) used to translate variable length items.

CLASS TEST(Item : item-type);
--(COBOL) used to perform class tests for variable-length items and those fixed-length items over 256 bytes long.

SEGMENTATION -- (COBOL) used to load segments of a program that are not in core storage and to pass control from one segment to the other.

STOP RUN -- (COBOL) controls exiting from the program and is entered when a program receives initial control.

SORT -- acts as an interface between the COBOL calling program and the Sort/Merge program.

ACCEPT(T : text);
-- (COBOL) is called to read from SYSIN or from the operator's console at execution time. For SYSIN, a logical record size of 80 is assumed.

CHECKPOINT -- (COBOL) is used when checkpoints are taken in a program.

DISPLAY(T : text);
-- (COBOL) is used to print, punch or type data usually in limited amount.

PERFORM(T : text);
-- (COBOL) is a call to a paragraph heading in COBOL. Execution is transferred to that paragraph name and the statements under that heading are executed.

PASCAL PREFIX

A set of procedure heading declarations and the necessary CONST and TYPE declarations. The prefix is added to the beginning of each Pascal program and is used to call any of the supporting routines.

MISCELLANEOUS ROUTINES

```
DATE(Date-string : Actualdate);
    -- returns the date in yymmdd format.

JULIANDATE(JDate-string : ActualJdate);
    -- returns the date in yy/ddd format.

TIME(Time-string : Actualtime);
    -- returns the time in  yymmddhh format.

CLOCK : integer;
    -- returns the number of milliseconds of central
    processor time used during execution of the program.

CLOCKLEFT : integer;
    -- returns the number of milliseconds of central
    processor time remaining to be used during execution.

WALLCLOCK : integer;
    -- an integer function of no arguments returns the time in
    seconds in 00:00:00 GMT

SEED(I : integer);
    -- the random number generator used to generate integers
    randomly.

TERMINAL(F : file-type) : boolean;
    -- returns true if the peripheral device associated with the
    file F is a terminal.

GETPARM(S : string-type, Len : integer);
    -- returns the program parameters supplied on the operating
    system command line or control card used to execute the
    program.

MESSAGE(S : string-type);
    -- writes the string S into the job execution log (the SYSMSG
    data set in OS and VS systems) or to the terminal in
    interactive systems.
```

```
HALT(S : string-type);
    -- terminates the program with a traceback and dump of local
    variables if the program was appropriately compiled.

SETCC(RC : integer);
    -- sets the completion code to the value RC.

TRAPPATTN(N : integer);
    -- indicates that N or fewer attention interrupts from the
    terminal should be counted but otherwise ignored.

SYSTEM(S : string-type, RC : integer);
    -- submits the string S to the host operation system to be
    executed as a command and stores the return code
    (condition code) from that command in RC.  The
    interpretation of the string S depends on the host operating
    system.

UNDEFINED(U : real) : boolean;
    -- a boolean function with a real number as it's argument;
    this function always returns false.
```

ERROR HANDLER

-- a routine designed to handle calls made to routines not contained within the common library. The error handle prints an error message and either terminates the process or enables the process to continue execution at the point after the error occurred depending on the severity of the error.

APPENDIX C

LIBRARY FUNCTIONS TO INVOKE SYSTEM PRIMITIVES

LIBRARY FUNCTIONS TO INVOKE UNIX SYSTEM PRIMITIVES.

ACCESS -- determines accessibility of a file.

ACCT -- enable or disable process accounting.

ALARM -- set a process's alarm clock.

BRK -- change data segment space allocation.

CHDIR -- change working directory.

CHMOD -- change mode of a file.

CHOWN -- change owner and group of a file

CHROOT -- change root directory.

CREAT -- create a new file or rewrite an existing one.

DUP -- duplicate an open file descriptor.

EXECL -- execute a file.

EXIT -- terminate a process.

FCNTL -- file control.

FORK -- create a new process.

GETPID -- get process, process group, and parent process IDs.

GETUID -- get real user, effective user, real group, and effective group IDs.

IOCTL -- control device that performs a variety of functions on character special files.

KILL -- send a signal to a process or a group of processes.

LINK -- link to a file.

LOCKING -- provide exclusive file regions for reading or writing.

LSEEK -- move read/write file pointer.

MKNOD -- make a directory or a special or ordinary file.

MOUNT -- mount a file system.

NICE -- change priority of a process.

OPEN -- open a file for reading or writing.

PAUSE -- suspend process until signal.

PHYS -- allow a process to access physical addresses.

PIPE -- create an interprocess channel.

PROFIL -- execution time profile.

PTRACE -- process trace.

READ -- read from a file.

REBOOT -- reboot the system.

SETPGRP -- set process group ID.

SETUID, SETGID -- set user and group IDs.

SIGNAL -- specify what to do upon receipt of a signal.

STAT -- obtain file status.

STIME -- set time.

SYNC -- update super-block.

TIME -- obtains the value of time in seconds since 00:00:00 GMT, January 1, 1970

TIMES -- gets process and child process times.

ULIMIT -- get and set user limits.

UMASK -- set file creation mode mask.

UMOUNT -- unmount a file system.

UNAME -- get name of current UNIX system.

UNLINK -- remove directory entry.

USTAT -- get file system statistics.

UTIME -- set file access and modification times.

UVAR -- returns system specific configuration information.

WAIT -- wait for child process to stop or terminate.

WRITE -- write on a file.

B I B L I O G R A P H Y

ACT PASCAL PROGRAM LOGIC MANUAL , Advanced Computer Techniques Corporation, New York, NY, (January 1983), pp. 6.1-6.11.

Administrator's Guide, THE UNIX SYSTEM, Issue 1, January 1983, pp. 121-127.

IBM OS FULL AMERICAN NATIONAL STANDARD COBOL COMPILER AND LIBRARY, VERSION 2 PROGRAMMERS GUIDE, International Business Machines Corp., New York, NY, (July 1972), pp. 277-81.

IBM SYSTEM/360 AND SYSTEM/370 FORTRAN IV LANGUAGE REFERENCE MANUAL Internation Business Machines, San Jose, CA (May 1974), pp. 117-23.

Joy, W. N. , S. L. Graham and C.B.Haley, BERKELEY PASCAL USER'S MANUAL VERSION 2.0, University of Berkeley Computer Center Library, Berkeley, CA (1980).

Morris, Robert and Ken Thompson, PASSWORD SECURITY: A CASE HISTORY Bell Laboratories, Murray Hill, NJ, p. 6.

NON-STOP SYSTEMS, TANDEM FORTRAN 77 REFERENCE MANUAL, Tandem Computers Inc., Cupertino, CA (October 1982), pp. 4.2.1-4.2.5.

Payne, Kendall R. PASCAL RUNTIME LIBRARY, Tandem Nonstop II, Manhattan, KS, (December 1983).

PERKIN ELMER PASCAL USER GUIDE AND LANGUAGE MANUAL AND RUNTIME SUPPORT REFERENCE MANUAL, Perkin Elmer Corp. - Computer Systems Division, Oceanport, NJ, (1980), pp. 10.27-10.32.

Perkin, Hal, PASCAL 8000 REFERENCE MANUAL/VERSION 2.0 Cornell University, USA (August 1980), pp. 45-55, 57-64.

Ritchie, Dennis M., ON THE SECURITY OF UNIX, Bell Laboratories, Murray Hill, NJ, pp. 1-3.

SYSTEM 3 UNIX OPERATION SYSTEM USER MANUAL, Vol 1 Unisoft Corp., Berkeley, CA (1983).

Thompson, K. UNIX IMPLEMENTATION, Bell Laboratories, Murray Hill, NJ, pp. 7-8.

UNIX SYSTEM USER'S MANUAL, SYSTEM 5, Western Electric Company (1983), Crash(8), pp. 1-3.

THE DEVELOPMENT AND ANALYSIS OF A PORTABLE RUNTIME LIBRARY ACCESSABLE TO ALL FORTRAN, COBOL AND PASCAL COMPILERS UNDER THE UNIX SYSTEM 5 OPERATING SYSTEM

by

KENDALL ROBERT PAYNE

B.S., Baker University, 1982

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY

Manhattan, Kansas

1984

ABSTRACT

This paper discusses and analyzes the development of a portable run time library that will support compilers for three different languages: COBOL, Fortran and Pascal. The compiler will run under the UNIX System-5 operating system.

The portable run time library will contain twelve different types of routines: input/output, file handling, storage management, arithmetic, logic, data transfer, control, the Pascal prefix, COBOL verb routines, the error handler and miscellaneous routines. Different groups of routines are applicable to different compilers. The I/O, file handling, storage management, data transfer, the Pascal prefix and the miscellaneous routines apply primarily to Pascal. The arithmetic and logic routines are common to both Pascal and Fortran while the COBOL verb routines, the control routines and the conversion routines relate to COBOL. The error handler routine applies to all three languages.

The routines in the run time library can be divided into two different categories according to their parameters. The two categories are functions and procedures. The routines that are functions are the arithmetic, logic and conversion routines. The group of routines classified as procedures can be subdivided into four groups: procedures with no parameters, one parameter, two parameters and multiple parameters. The initialization, utility, control and error handling routines are all procedures with no parameters. All input/output routines and most of the file handling routines will have one or more parameters depending on the nature of the rou-

time. The storage management and data transfer routines all have two or more parameters.

The library routines in the UNIX system 5 run time library have very little in common with routines in the portable run time library. The routines in the Unix library: maintain the file system, create and destroy processes, keep a log of all processes currently running on the system and keep a timetable of all running processes. On the other hand, the routines in the portable run time library are tailored to perform functions for the individual user.

There are four important features that should be taken into consideration when determining whether the UNIX operating system can operate in a portable environment or not. They are security, system recovery, file integrity and overall performance. These features also play an important role in maintaining the integrity of UNIX and determining whether the interfaces between UNIX and the portable run time library function properly.

The primary functions of the run time library are to provide flexibility in allowing users to write software in a portable environment using Fortran, COBOL and Pascal and to allow an individual to interface with a wide variety of different routines.