1563

Bell Laboratories Cover Sheet for Technical Memorandum

The information contained herein is for the use of employees of Bell Laboratories and is not for publication (see GEI 13.9-3)

Title: **Cycles to Burn: The Challenge of Abundance**

Date: **July 3, 1980**

Other Keywords: **UNIX**

TM: **80-3633-5**

Author(s)	Location	Extension
Alan R. Feuer	**MH 2C-273**	**3136**

Charging Case: **49343-12**
Filing Case: **49343-12**
3633-800703.01TM

ABSTRACT

The computing power available per person continues to grow rapidly. This paper looks at some ways that the increased power will change what we expect a computing system to do for us. Some deficiencies of UNIX™ are used as a vehicle for exploring these new expectations.

Pages Text: 4 Other: 1 Total: 5

No. Figures: 0 No. Tables: 0 No. Refs.: 3

BELL TELEPHONE LABORATORIES, INC. TM-80-3633-5

DISTRIBUTION
(REFER GEI 13.3-3)

COMPLETE MEMORANDUM TO

CORRESPONDENCE FILES

OFFICIAL FILE COPY
PLUS ONE COPY FOR
EACH ADDITIONAL FILING
CASE REFERENCED

DATE FILE COPY
(FORM E-1328)

10 REFERENCE COPIES

ALDERSON,ANGELA L
ANDERSON,MILTON M
ANSELMO,DONALD R
ARNOLD,THOMAS F
BARSKY,ALLEN
BAYER,C L
BECKER,J
BECKETT,J T
BERGLAND,G D
BIREN,EMMA B
BLAHUT,D J
BLUM,MARION
BOEHM,EARL W
BOHNING,J E
BOWYER,L RAY
BOYCE,W M
BROWN,ELLINGTON L
BROWN,W STANLEY
BURNETTE,W A
BUTLETT,RUSSELL L
<CANADAY,HUGO H
CERMAK,I A
CHANG,HERBERT T
CHELLIS,ALICIA L
CHEN,STEPHEN
CHEN,T L
CHERRY,LORINDA L
CHI,HSI-OAO
CHONG,PHEE
CHOW,W F
COOK,T J
COPP,DAVID H
DE GRAAF,J A
DE LUGISH,B G
DICKMAN,BERNARD N
DOLOTTA,T A
DOMPLERAS,J A
DRISCOLL,P J
DWORAK,F S
FABRICIUS,WAYNE N
FAULKNER,EDGAR A
>FELDMAN,STUART L
FEUER,ALAN R
FISHMAN,DANIEL H
FOUGHT,E J
FRASER,A G
FREDERICKS,A A
FREEMAN,K G
FREEMAN,MARTIN
FROST,B BONNELL

GALLAHER,L E
*GEHANI,NARAIN H
GEORGEN,MICHAEL E
GEYLING,F T
GIBB,KENNETH R
GIBSON,H T,JR
GILKEY,THOMAS J
GRAHAM,R L
GRAVEMAN,R F
GREENWOOD,T S
GUIDI,PIER T
HAGGERTY,JOSEPH F
HAIGHT,R C
HALL,ANDREW J,JR
HAMILTON,PATRICIA A
HANSEN,GWEN J
HENIG,FRANCES B
HERMAN,KENNETH M
HOFMANN,A M
HOLTMAN,JAMES F
HOLTZMAN,JACK M
HO,TSIEN-LIN
IRVINE,M M
JACOBSEN,ROBERT C
KANODIA,RAJENDRA K
KEESE,W M
KERNIGHAN,BRIAN W
KIRSLIS,PETER A
KOSMAN,ROBERT A
KRETSCH,KENNETH F,JR
LANGER,JEFF
LAUTENBACH,DEBORAH A
LAYTON,R J,JR
LEE,JOHN R
LEVENBERG,R A
LINDERMAN,JOHN F
LOIKITS,E A
LUDERER,GOTTFRIED W R
MACIASZEK,JOHN T
MARIONE,JOHN F
MARSHALL,WILLIAM T
MASTELLOTTO,M A
MARTERSTECK,K E
MATHIOT,MARTIN J
MC DONEGAL,CAROL A
MC LAUGHLIN,J
MENNINGER,E E
MICHAELS,ANN S
MOLINELLI,JOHN J
MORGAN,DENNIS J
NOWITZ,D A
OH,RICHARD YOUNG
PEILGRIM,J F
<PHILLIPS,E J
PINSON,ELLIOT N
PRESOTTO,DAVID L
PRITZE,BARTON G
PUEBLING,BRUCE F
QUINN,MARGARET E
RALEIGH,T M
REED,A A
RIDDLE,GUY G

ROBERTS,CHARLES S
ROBIGUZZ,ERNESTO J
ROME,W D
*ROSIN,ROBERT FISHER
ROSLER,LAWRENCE
ROYEGNO,EILEN D
SARSEVITZ,A L
SATZ,LAWRENCE E
*SCANLON,J M
SCHMITT,C A
SCHUSTER,PAUL A
SEEGDA,SCHDAN
*SHARMA,D K
SHORTER,J F
SIMONE,C F
SINGH,S P
SINOWITZ,NORMAN E
SLANA,M F
<SMITH,M T,JR
SO,E C
SPIESS,E J
STROHECKER,GARY A
TARLOSKI,THEODORE F,JR
TAGUE,BERKLEY A
THOMAS,LEE C
THOMPSON,A
TOY,W N
<ULLRICH,RITA T
VEERARAG,G M
WALKER,ELIZABETH A
WANDZILAK,PHILIP D
WANG,DA-LUN
WATSON,C S
WATTERS,RICHARD J
WETHERELL,CHARLES S
WILSON,D E
YIOOBELLIS,ROBERT E
YU,JOHN L
150 NAMES

COVER SHEET ONLY TO

CORRESPONDENCE FILES

4 COPIES PLUS ONE
COPY FOR EACH FILING
CASE

AARONSON,STEVE M
ABATE,JOSEPH
ACKERMAN,A FRANK
ACKERMAN,J T
AHO,ALFRED V
AHRENS,WALTER E
ALCALAY,D
ALLEN,JAMES E
ALLISON,C E,JR
ALMQUIST,R F
AMORY,ROBERT W
AMOSS,JOHN J

ANDERSON,KATHRYN J
ARNOLD,GEORGE W
ASELTINE,EDWARD G
ASTHANA,ABHAYA
ATAL,BISHNU S
BAILEY,CATHERINE T
BAKER,MITCHELL E
BALDWIN,GEORGE L
BARBATO,ROBERT E
BARON,ROBERT V
BARRESE,A L
BARR,W J
BAUER,BARBARA T
BAUER,E C
BECKER,RICHARD A
BENCO,DAVID S
BENISCH,JEAN
BENNETT,RAYMOND W
BENNETT,RICHARD L
BENSING,JAMES EDWARD
BERENBAUM,ALAN
BERNHARDT,RICHARD C
BERNSTEIN,L
BEYER,JEAN-DAVID
BICKFORD,NEIL E
BILOWOS,E M
BISHOP,THOMAS F
BITTNER,E E
BLAKE,GARY D
BLAZIER,S D
BLEICHER,E
BLEIER,JOSEF
BLINN,J C
BLOSSER,PATRICK A
BODEN,F J
BOGART,THOMAS G
BOIVIE,RICHARD E
BORKAR,SUBESH E
BOSE,DEBASISH
BOURNE,STEPHEN E
BOYER,PHILLIS J
BOYLE,GERALD C
BRADLEY,M HELEN
BRANDT,RICHARD E
BRAUN,DAVID A
BREITHAUPT,A E
BROAD,MARTHA M
BROSS,JEFFREY D
BROYMAN,INNA
BROWN,LAURENCE MC FEE
BUCZ,S C
BUZICKER,DOUGLAS E
BURGESS,DIANE MARIE
BUSG,F M
BUSIC,MILORAD E
BUSOFF,STEVEN J
BUZZINSKI,N J
BUTTON,BEVERLY
BYRNE,EDWARD E
CABLE,GORDON G,JR
CAMPBELL,JERRY E
CARTER,DONALD H

CISPERS,BARBARA E
CITO,E E
CIVINESS,JOHN D
CHAPPEL,N F
CHAI,D T
CHAMBERS,B C
CHAMBERS,J M
CHANDRA,SARIRE
CHANG,JO-MEI
CHANG,S-J
CHENG,T
CHEN,PING C
CHIANG,T C
CHIEN,ANTHONY E
CHILDS,CAROLYN
CHIN,AUGUSTIN T
CHODROW,M M
CHRIST,C W,JR
CHU,NARISA N
CHU,PAUL E N
CICHINSKI,DEBRA F
CIRILLO,C
CLAYTON,D F
CLOUTIER,J
COATES,KAREN E
COHEN,HARVEY
COLE,LOUIS M
COLE,MARILYN C
COLLICOTT,E E
COLTON,JOEN E
CONKLIN,DANIEL L
CONNERS,RONALD E
COOPER,MICHAEL E
COVINGTON,RALPH L
CRAGUN,DONALD W
CRISTOFOR,EUGENE
CRUME,L L
CRUPI,JOSEPH A
DALRYMPLE,FREDERICK L
DAVIDSON,CHARLES LEVI
DAVIS,E DREW
DE PAZZO,M J
DEAN,JEFFREY S
DENNY,MICHAEL S
DI PIETRO,E S
DINEEN,THOMAS J
DIYARBAKUNI,E E
DOLATOWSKI,VIRGINIA M
DOWDEN,DOUGLAS C
DOWDEN,IRIS S
DREITZER,A K
DUDICK,ANTHONY
DUGGER,DONALD D
DUNCANSON,ROBERT L
EDMUNDS,T W
EELINGER,JAMES C
EISEN,STEVEN E
EITZLBACH,DAVID L
ELDRIDGE,JOHN
ELY,T C
ENGLAR,BRUCE WYATT
EPLEY,ROBERT V
...
561 TOTAL

* NAMES BY AUTHOR > CITED AS REFERENCE < REQUESTED BY READER (NAMES WITHOUT PREFIX
WERE SELECTED USING THE AUTHOR'S SUBJECT OR ORGANIZATIONAL SPECIFICATION AS GIVEN BELOW)

MERCURY SPECIFICATION.....

COMPLETE MEMO TO:
363-SUP 3646-SUP 3644-SUP 36-DRB 36-DIR 3633

COCS# = COMPUTING SYSTEMS, PANORAMIC OR ECCENTRIC DOCUMENTS ONLY

COVER SHEET TO:

MC CORRESPONDENCE FILES TM-80-3633-5
MC 81127 TOTAL PAGES 5

TO GET A COMPLETE COPY:

1. BE SURE YOUR CORRECT ADDRESS IS GIVEN ON THE OTHER SIDE.
2. FOLD THIS SHEET IN HALF WITH THIS SIDE OUT AND STAPLE.
3. CIRCLE THE ADDRESS AT RIGHT. USE NO ENVELOPE.
4. INDICATE WHETHER MICROFICHE OR PAPER IS DESIRED.

PLEASE SEND A COMPLETE

( ) MICROFICHE COPY ( ) PAPER COPY

TO THE ADDRESS SHOWN ON THE OTHER SIDE.

Bell Laboratories

subject: **Cycles to Burn: The Challenge of Abundance**
Case: **49343-12**
File: **49343-12**

date: **July 3, 1980**

from: **Alan R. Feuer**
MH 3633
2C-273 x3136
3633-800703.01TM

TM: **80-3633-5**

MEMORANDUM FOR FILE

1. INTRODUCTION

We are entering an era in which computing power will become a plentiful resource. Some predict that within four years the computing power of a VAX 11/780, or even an IBM 370/158 [2], will commonly be available per terminal (and perhaps inside the terminal). This abundance challenges conventional views of what a computing system is.

Most existing computing systems were designed for environments where computing power is scarce. UNIX™, for example, was designed to share a quite small computer among several users. As the power available per person grows we will come to expect services from our computing systems that are not possible in today's environments.

What are these new expectations likely to be? To answer this question I have taken a critical look at the UNIX system. UNIX has deservedly enjoyed great success. There is much within its design philosophy that should be inherited by future systems. But there are areas where UNIX has problems. It is from such problems that new expectations arise.

To most users, the *system* is the computing model presented by the outermost layer of programs. In UNIX this is the model presented by the ideas in Section 1 of the User Manual [3], principally the Shell and the file system. I chose to examine the UNIX system because it meets so many of our current expectations so well. It is a good example of what can be achieved given the computing power available today. Most systems suffer from the deficiencies of UNIX plus a host of others.

This paper is mostly a discussion of problems. No real solutions are presented. My intent is to encourage thinking about what comes next, even at this early stage when traditional timesharing still serves us well. Read on, critically.

2. THE NAMING PROBLEM

A typical thing one does on a general purpose computing system is to create objects, such as memos, letters, pictures, and programs. In order to save an object for later access UNIX requires that the object be stored in a file. In order for the file to be unambiguously accessible, the file must be given a unique (full) name. This has two unfortunate effects.

First, one is forced to invent (and remember) symbolic names for objects that do not really have names. Temporary files are sometimes in this group. Letters, pictures, and memos are almost always in this group. Thus one accesses "the letter last year to the Berkeley Computer Science Division" with something like "letters/berkeley79.csd2," and "last week's plot of log

response time versus the number of active processes" with something like "perf/response/procs.6-23" (or was it "perf/6-23/response.procs").

Second, one is forced to invent uncommon names for common entities. As an example consider some of names given to the generic command **print** in PWB/UNIX [3]:

- prs, prt: print an SCCS file
- man: print a manual page
- pib: print a programmer information bulletin
- cat: print text unpaginated and unformatted
- pr: print text paginated and unformatted
- nroff, mm: print text paginated and formatted

Or consider the names that must be given to the logical parts of an object, for example a program. The source code resides in a file of one name, the object code lies in another file of another name, the executable in still another file with a different name, and the documentation in yet another file with yet a different name. (You may argue that if one knows the conventions these various names are not troublesome. Okay, then where is the source for that most common of commands cd? And once you've found it how does it help you find its documentation?)

The symptom of the naming problem is that we are forced to create too many names. Our directories are full of files with names whose meaning we have forgotten and we call upon generic commands with an array of less than telling aliases.

3. THE ACCESS PROBLEM

The access problem is related to the naming problem. All objects are stored in files and accessed by the file name. While it is true that a logical file system, such as on UNIX, frees users from having to know which disk drive a file is stored on, or even whether or not it is stored on a disk, these systems still rely on file names which are really just symbolic addresses. That is, objects are accessed by knowing where they are stored.

To illustrate the problem of access by location consider the task of finding the source code for a particular system function. Suppose someone suggests that you look at the standard library routine *fwrite*.

You might begin by looking in your programmer's manual[1] for a writeup on *fwrite*. You thumb immediately to Section 3 without going to the index because you know that standard library routines are located there. To your befuddlement there is no page for *fwrite*. "Aha," you say, "it must be out of order." But looking a couple of pages each way doesn't bring much success. You decide to check the index after all and discover that the writeup for *fwrite* is stored under *fread*.

After reading about *fwrite*, you decide to look at the source code. Never having looked at system code before you begin your search from /. A few false starts looking under *sys* and *unix* lead you to the unlikely candidate *usr*. *Usr* has three reasonable choices. After you've searched through *lib* and *pub* (and ruled them out) you look into *src*. *Src* looks good as it has the subdirectory *lib*. But within *lib* nothing looks good. Recalling the classification of libraries from Section 3 of the manual you decide to rule out the directories that you know: *libF77* is the fortran library, *libPW* is the PWB library, *libc* is the C library, *libm* is the math library, *libplot* must be the plot library. It must be in *libl* or *liby*. Quick inspection shows that it is in neither. Your next guess was *libc*, which looks very good as it has a subdirectory *stdio*. You descend into *stdio* and cat *fwrite.c*. But *fwrite.c* is not to be found. Then it strikes you. "Of course, since the writeup for *fwrite* was stored under *fread*, the source for *fwrite* must be stored under *fread.c*."

1. The *Programmer's Workbench* manual [3] and system are used for this example.

Again you are wrong. So you ls the directory to look for something close. But none of the file names are close, though there are a lot of *fnames*: *fget*, *fopen*, *fseek*, etc. Tired and weary, you decide to grep for *fwrite* hoping not to be overwhelmed with usages. And at last, you find it in *rdwt.c*.

Amazed at how long it took to find *fwrite* by manual searching (and realizing that not even an exhaustive search using **find** would have helped) you muse that someone has certainly written a command to do just what you've done. In fact you can recall mention of such a command, it was called **findCfunc**. But you don't remember where **findCfunc** was located. So you decide to look for it.

You might begin by looking in your programmer's manual for a writeup on **findCfunc**. ...

The heart of the problem is that we usually are interested in what an object *is* rather than in what an object is called. Calling a rose *Rosa odorata* or "that flower" subtracts nothing from its fragrance. Many of the characteristics desirable for accessing the records of a data base, such as multiple access paths and access by contents, are desirable for accessing the objects of a computing system.

4. THE UPDATE PROBLEM

A difficult problem in maintaining a collection of interrelated parts is to know which part depends upon which others, that is, which parts are affected by a change in some other part. An example of this in programming would be to determine which in a collection of modules must be recompiled as a result of a change in one of the modules. The **make** [1] program addresses the update problem.

Knowing how the parts of a whole are related is a problem on UNIX because the files in which the parts are stored are related only casually. In the best case the files are related by naming conventions, but often they are related only by knowledge of their contents. For this reason **make** requires explicit instructions as to how the parts fit together. Even with the knowledge of how the files depend upon one another, **make** at best is too conservative. It has no way to know if a change in one file requires rebuilding another. For example, *prog.c* may depend only upon some of the definitions in *head.h*. Nevertheless, **make** will cause *prog.c* to be recompiled whenever any change is made to *head.h*.

Basically the problem is that dependencies are based on files and files are too coarse. We tend to collect many structurally complete objects within a single file. We do so partly for efficiency, to reduce the number of file accesses. We also pack many objects into one file because if we tried to store objects one per file our directories and our minds would soon be swamped with names.

5. THE COMPLEXITY PROBLEM

One likely scenario of a future computing environment is that within each terminal will be an operating system running a collection of cooperating concurrent processes. For example, while entering English text at a terminal, perhaps a formatter, a spelling checker, and a style enforcer might all be examining the input. An important question for such a scenario is how to manage the complexity, both internally (within each process) and externally (to the system user).

The Shell makes it easy to specify simple parallelism using pipelines. But pipelines are not sufficiently general given the abundant computing resource at hand. Also, they require too much information from the user, namely, in what order the data is to pass from process to process (should the spelling checker get a word after the formatter and before the style enforcer or after the style enforcer and before the formatter, or do I care?) One way to relieve the user of these decisions and still preserve the independence of the processes that pipes so nicely permit is to create shell procedures that set up the pipelines. as is done by commands such as **mmt**. The disadvantage to this approach is that such procedures must have prodigious option lists to allow some control over the internal commands, and the procedure names add to the naming problem by further cluttering the name space for commands.

It seems that two devices are needed. The first allows the construction of networks of programs with the same ease one constructs pipelines. The second allows the request of services without having to specify how the underlying programs interact. One should be able to ask for "this text to be formatted as a technical memo, output on the typesetter, with reference, table, picture, and equation processing as needed."

6. THE LANGUAGE PROBLEM

The evolution of UNIX facilities is evident in the variety and quantity of languages a user must learn. This has been praised in that one only needs to learn the languages for the facilities one wishes to use. Also, this allows the languages to match the application better than some general language to be used in all domains. The unfortunate consequence is that the different languages often have different and sometimes conflicting conventions.

For example, consider the variety of command line formats. Options are *usually* introduced by a leading minus, but some commands require them to be specified separately and some require them to be specified together. Sometimes the options can go anywhere on the command line and sometimes they must follow the command name. A sole minus as an argument sometimes references the standard input and sometimes it is itself a command option.

Or consider the construction of regular expressions for matching text. A good deal of effort has been spent to unify the syntax across commands. Nevertheless, ed and sh, two of the most used commands, have different conventions. Or consider the variety of forms for the IF-construct found in commonly used programs. C has one form, the C preprocessor has another, the Shell another, and nroff/troff still another.

What is missing is a model for common constructs. Thus far, no form of the basic constructs has managed to hold the attention of system program designers. The forms in C have come closest, reappearing in bc and in awk. Still, most programs remain individualistic, including the recently rewritten Shell.

7. CONCLUSION

This has not been an easy paper for me to write. I owe much to UNIX for raising my standards of what to expect from a computing environment. But the world is not stationary, not the hardware, not the software, and not our imaginations. UNIX is not getting poorer, we are getting richer.

The problems I have discussed will not go away with more cleaning and fine tuning of UNIX. A simple change, like making regular expressions in the Shell more like those in the editor by decreeing "." shall replace "?" to match any single character, would surely cause paranoia at UNIX terminals. Though strictly such a problem does not belong to the operating system, it and many like it are a consequence of decisions spread throughout the system.

The challenge of abundance is to conceive of a computing model that has the elegance of UNIX plus the powerful interface we will come to expect when one has cycles to burn.

8. ACKNOWLEDGEMENTS

I am grateful to Rudd Canaday, Dan Fishman, Narain Gehani, John Linderman, and Bill Roome for their comments on an earlier draft of this memorandum, and to Charles Wethereil for his incessant insight.

Alan Feuer

MH-3633-ARF-unix

Alan R. Feuer

Att.
References

9. REFERENCES

[1] Feldman, S. I.; "Make—A Program for Maintaining Computer Programs", Bell Laboratories.

[2] Hnatek, E. R.; "Semiconductor Memory Update—Part 1: ROMS", *Computer Design*, vo. 18, no. 12, December 1979.

[3] *PWB/UNIX User's Manual—Release 2.0*, Bell Laboratories, June 1979.