HO

1545

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The information contained herein is for the use of employees of Bell Laboratories and is not for publication (see GEI 13.9-3)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: **Kernel Profiling for UNIX Systems on the VAX-11**

Date: **March 11, 1980**

Other Keywords: **performance measurement**

TM: **80-1353-3**

Author(s)	Location	Extension	Charging Case: 39394
John F. Reiser	**HO 4F635**	**3942**	Filing Case: **39394-21**

ABSTRACT

This paper describes a low-cost technique for constructing an accurate profile (histogram of sampled values of the program counter, aggregated to the procedure level) for the system kernel of a UNIX-based operating system running on the VAX-11 computer. The method also reduces the uncertainty associated with previous methods of aggregating the sampled values. (The spatial quantization error has been reduced to essentially zero.) The technique can be adapted to other machines and to code running in user mode, with the added benefit that it can yield meaningful profiles for code running under a software interpreter. An appendix includes an example of the technique applied to tuning the UNIX/32V system on a VAX-11.

Pages Text: 4	Other: 6	Total: 10	
No. Figures: 0	No. Tables: 3	No. Refs.: 1	

BELL TELEPHONE LABORATORIES, INC.

TM-80-1353-3

DISTRIBUTION
(REFER GEI 13.9-3)

COMPLETE MEMORANDUM TO

CORRESPONDENCE FILES

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

DATE FILE COPY
(FORM 2-1328)

10 REFERENCE COPIES

ALLES,H G
<BARON,ROBERT V
BERGLAND,G D
<CHOW,MING-CHWAN
<CHRISTENSEN,C
CLOGSTON,ALBERT M
FREENY,STAN I
+HANNAY,N B
<KEESE,W M
KOGELNIK,H
LIMB,JOHN O
LUCKY,R W
<MC DONALD,H S
+MC DONNELL,J P
<MENNINGER,R E
MILLER,STEWART E
NETRAVALI,A N
NINKE,WILLIAM H
PENZIAS,A A
+PRIM,R C
REUDINK,D O
ROBERTS,CHARLES S
<ROSLER,LAWRENCE
SLICHTER,W P
<STORER,JAMES A
<SZYMANSKI,THOMAS G
<TEWKSBURY,S K
27 NAMES

COVER SHEET ONLY TO

CORRESPONDENCE FILES

4 COPIES PLUS ONE
COPY FOR EACH FILING
CASE

AAGESEN,JOHN
ABATEMARCO,TERESA M
ABATE,JOSEPH
ACKERMAN,A FRANK
ACKERMAN,J T
AHO,ALFRED V
AHRENS,RAINER B
<AHUJA,SUDHIR R
ALBAGLI,V R
ALBERTS,BARBARA A

COVER SHEET ONLY TO

ALCALAY,D
ALEXIS,A D,JR
ALKONS,FREDERICK
ALLEN,JAMES J,JR
ALLEN,JAMES R
ALLISON,C E,JR
AMABILE,GEORGE R
AMES,WILLIAM E
AMITAY,N
AMOSS,JOHN J
ANDERSON,C R
ANDERSON,KATHRYN J
ANDERSON,L G
ANDERSON,MILTON M
ANDERSON,R E
ANDERSON,R R
ANDREWS,W J
ANTOLICK,DAVID R
ANTONELLI,CHARLES J
ARCHER,RUSSELL E,JR
ARMSTRONG,F O,JR
ARNOLD,GEORGE W
ARNOLD,JAMES Q
ARNOLD,PHYLLIS A
ARNOLD,THOMAS F
ARTHURS,E
ARTIS,H P
ARVIDSON,W P
ASELTINE,EDWARD G
ASMUTH,RICHARD L
ASTHANA,ABHAYA
ATAL,BISHNU S
AXELSON,A L
BACCASH,JEANNE M
BACH,MAURICE J
BACKUS,C F,SR
BAGLEY,JOHN D
BAILEY,CATHERINE T
BAKER,DONN
BAKER,MITCHELL B
BALDWIN,GEORGE L
BALENSON,CHRISTINE M
BALLANCE,ROBERT A
BANG,SUNG YANG
BARAN,S J
BARBATO,ROBERT R
BARNHARDT,KARL R
BAROPSKY,ALLEN
BARRESE,A L
BARR,DAVID L
BATISTONI,F J
BATTAGLIA,FRANCES
BAUER,BARBARA T
BAUER,H C
BAUER,HELEN A
<BAUER,WOLFGANG F
<BAUGH,C R
<BAXTER,LESLIE A
BAYER,D L
BECKER,JACOB I
BECKER,RICHARD A

COVER SHEET ONLY TO

BECKER,SHELDON T
BECKETT,J T
BEIGHLEY,KEITH A
BELANGER,DAVID G
BENCO,DAVID S
BENISCH,JEAN
BENNETT,RAYMOND W
BENNETT,RICHARD L
BENNETT,WILLIAM C
BENOWITZ,P
BENSING,JAMES EDWARD
BERENBAUM,ALAN
BERGERON,R F,JR
BERKEY,M A
BERK,DONALD A
BERNHARDT,RICHARD C
BERNOSKE,BEVERLY G
BERNSTEIN,DANIELLE R
BERNSTEIN,L
BERNSTEIN,PAULA R
BERZINS,ALEXANDER H
<BEYER,JEAN-DAVID
BHATIA,RAJIV
BIANCHI,M H
BICKFORD,NEIL B
BILASH,TIMOTHY D
BILONOS,R M
BIREN,IRMA B
BISHOP,J DANIEL
BISHOP,THOMAS P
BITTNER,B B
BITTRICH,MARY E
BLACHMAN,NANCY
BLAKE,GARY D
BLAZIER,S D
BLEIER,JOSEF
BLEWETT,C DOUGLAS
BLINDER,S
BLINN,J C
BLOSSER,PATRICK A
BLUM,MARION
BOCKUS,ROBERT J
BOCK,NANCY E
BODDIE,JAMES R
BODEN,F J
BOEHM,BARL W
BOEHM,KIM R
BOGART,F J
BOGART,THOMAS G
BOIVIE,RICHARD M
BOLICK,LAWRENCE J
BOLINSKI,NANCY V DEVLIN
BONACHEA,R N
BONANNI,L E
BOND,F C
BOND,HOLTON C,JR
BORG,KEVIN E
BORISON,ELLEN A
BOSE,DEBASISH
BOSTON,RONALD E
BOSWELL,PAULA S

COVER SHEET ONLY TO

BOTHUN,R H
BOURNE,STEPHEN R
BOWYER,L RAY
BOYCE,W M
BOYER,PHYLLIS J
BOYLE,GERALD C
BRADFORD,EDWARD G
BRADLEY,M HELEN
BRANDAUER,C M
BRANDT,RICHARD B
BRAUN,DAVID A
BREILAND,JOHN R
BRENSKI,EDWIN F
BRESLER,RENEE A
BRITT,WARREN D
BROAD,MARTHA M
BRONSTEIN,N
BROOKS,CATHERINE ANN
BROSS,JEFFREY D
BROVMAN,INNA
BROWN,C W
BROWN,ELLINGTON L
BROWN,LAURENCE MC FEE
BROWN,MARK S
BROWN,STUART G
BROWN,W R
BROWN,W STANLEY
BROZ,G C
BRUECKNER,DOUGLAS E
BRYANT,DAVID J
BUCK,I D
BURGESS,DIANE MAHER
BURGESS,JOHN T,JR
BURG,F M
BURIC,MILORAD R
BURKE,R J
BURNETTE,W A
BURNETT,DAVID S
BURNET,ROSE M
BUROFF,STEVEN J
BURROWS,THOMAS A
BUTLETT,DARRELL L
BUTTON,BEVERLY
<BUTZIEN,PAUL E
BYORICK,ROBERT S
BYRNE,EDWARD R
CALL,PETER F
CAMPBELL,JERRY H
CAMPBELL,MICHAEL R
<CANADAY,BUDD H
CANDREA,RONALD D
CANNATA,PHILIP E
CAREY,J E
CARRAN,JOHN H
CARR,DAVID C
CARR,RICHARD G
CARTER,DONALD H
CASEY,J L
CASPERS,BARBARA E
CASTELLI,LEONARD P
CATO,H E

COVER SHEET ONLY TO

CAVINESS,JOHN D
CERMAK,I A
CHAFFEE,N P
CHAIKIN,JAMIE A
CHAI,D T
CHAMBERS,B C
CHAMBERS,J M
CHANG,G K
CHANG,HERBERT Y
CHANG,JO-MEI
CHAO,C
CHAPMAN,LESLEY A
CHAPPELL,S G
CHARLES,JOYCE
CHEE,T
CHELLIS,ALICIA L
CHENG-QUISPE,ENRIQUE
CHENG,Y
CHEN,PING C
CHEN,ROBERT
CHEN,STEPHEN
CHEN,T L
CHERRY,LORINDA L
CHEUNG,ROGER C
CHE,HER-DAW
CHIANG,T C
CHIEN,ANTHONY H
CHILDS,CAROLYN
CHODROW,M M
CHONG,PHEE
CHRISTENSON,DENNIS A
CHRIST,C W,JR
CHUNG,TSUNG-JEN BEN
CHU,NARISA N
CHU,PAUL H N
CICHINSKI,STEVEN
CIEMINSKI,DEBRA F
CIESLAK,THOMAS J
CLARK,DAVID L
CLAYTON,D P
CLEWELL,JAMES L
CLIFFORD,COURTENAY B
CLINE,LAUREL M I
CLINE,TERRY W
CLOUTIER,J
COATES,KAREN E
COFFEY,JOHN R
COHEN,AARON S
COHEN,DAVID
COHEN,HARVEY
COHEN,KAREN L
COLEMAN,ELANE
<COLE,LOUIS M
COLE,MARILYN C
COLLICOTT,R B
COMRADE,GERALDINE
CONNERS,RONALD E
COOK,T J
COOMER,STEPHEN D
COOPER,MARTIN A
COOPER,MICHAEL H
...
1387 TOTAL

+ NAMED 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:
135-DPH 13-DIR 11-EXD 13-EXD 15-EXD 16-EXD

COVER SHEET TO:
COCSPM = COMPUTING SYSTEM PERFORMANCE MEASUREMENT, SIMULATION, ACCOUNTING
UNOS# = UNIX OPERATING SYSTEM: GENERAL OR SURVEY DOCUMENTS

REISER,J F
BO 4F635

TM-80-1353-3
TOTAL PAGES 10

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.

PLEASE SEND A COMPLETE COPY TO THE ADDRESS SHOWN ON THE OTHER SIDE
NO ENVELOPE WILL BE NEEDED IF YOU SIMPLY STAPLE THIS COVER SHEET TO THE COMPLETE COPY.
IF COPIES ARE NO LONGER AVAILABLE PLEASE FORWARD THIS REQUEST TO THE CORRESPONDENCE FILES.

Bell Laboratories

subject: **Kernel Profiling for UNIX Systems on the VAX-11**
Case: **39394**
File: **39394-21**

date: **March 11, 1980**

from: **John F. Reiser**
HO 1353
4F635 x3942

TM: **80-1353-3**

MEMORANDUM FOR FILE

1. Introduction

Periodically sampling the value of the program counter and constructing a histogram of the sampled values constitutes a useful tool for performance measurement of computer programs. The histogram often identifies where implementation or algorithm changes would have the largest effect on running time, and can also help a programmer understand the general operation of a program. The *prof* system call implemented in UNIX systems directs the operating system to sample the program counter of a user program 60 times per second and automatically accumulate the histogram.

Constructing a profile of the operating system itself is trickier. The system has an intimate relationship with the timer (there frequently is only one). The timer interrupt may be locked out intentionally to guarantee mutual exclusion when accessing certain data structures. The priority at which interrupts are serviced may be fixed by the hardware.

This paper describes a low-cost technique which profiles the kernel of a UNIX-based operating system running on a Digital Equipment Corporation VAX-11 computer. The method also reduces the spatial quantization error to essentially zero. The technique can be adapted to other machines and for use by code running in user mode.

2. Old Profiling Method for User Programs

An understanding of the existing profile mechanism of UNIX systems will prove helpful. The mechanism provides a histogram of sampled values of the program counter (usually aggregated to the procedure level), frequency counts of the number of times each procedure was called, and average time per call. Consider first the number of calls to each procedure. When invoked via *cc -p*, the compiler inserts code equivalent to

```
static long *pnc = 0;
mcount(&pnc);
```

at the beginning of each procedure. On the first call from each procedure *mcount* allocates a long integer counter from a private buffer, stores the address of the counter in *pnc*, associates the counter just allocated with the address to which *mcount* will return, and sets the counter to 1. On subsequent calls *mcount* increments the counter to which *pnc* points.

Next consider the histogram of sampled program counter values. The sampling and tallying is performed entirely by the operating system. The user specifies a tally array and a linear transformation which maps an interval of program counter values onto the tally array. (By default the domain of the transformation is the set of addresses occupied by instructions when

execution begins, and the profiling startup routine allocates the tally array so that the leading coefficient of the transformation is approximately 1/8.) When the clock ticks (every 1/60 second), the system takes the interrupted program counter, applies the transformation, and increments the appropriate cell in the tally array.

Files loaded with *cc -p* get a special startup routine, to which the operating system first transfers control when the program is executed. The startup routine allocates memory for the tally array and for the entrance counters maintained by *mcount*, requests that the operating system initiate program counter sampling, and calls the user's *main* procedure. When *main* returns, the startup routine terminates program counter sampling and writes the tally array and the *mcount* counters and return addresses to a file.

The *prof* program matches the entrance counts and tally cells from the file with the procedure names and entry addresses in the symbol table of the original program. There can be no ambiguity in associating the entrance counts with the right name because the compiler generates exactly one call to *mcount* for each procedure compiled under *cc -p*. However, *prof* must apply a heuristic to apportion the tallies from a cell whose preimage under the linear transformation contains portions of more than one procedure. (Many procedures could conceivably map to the same tally cell; but the leading coefficient and the procedures themselves are usually large enough so that the problem arises only with portions of two adjacent procedures.) *Prof* assumes a linear relationship between size of the preimage which maps into the tally cell and allotment of tallies. Together with the usually noninjective linear transformation, *prof's* assumption introduces a spatial quantization error in the reported time distribution.

3. New Profiling Method for UNIX Systems on the VAX-11

The basic strategy consists of speeding up the normal timer by a small factor (the factor 5 was used here), changing the compiled code (with *sed* run over .s files) to properly account for *spl6* timer lockouts, changing the compiled code (with *sed* run over *cc -p -S* files) to use the VAX-11 stack frame in a new way, and carefully coding the timer interrupt routine for fast execution.

The interval between timer interrupts was reduced by a factor of 5. At each interrupt the profiling code is executed. Every fifth interrupt the old clock scheduling routine is invoked. This scheme gives 300 samples per second while maintaining the old frequency of timer-driven actions.

The interval timer has an interrupt priority higher than that of any input/output device, so device driver routines would be profiled automatically except for raising their priority level to achieve mutual exclusion by locking out the timer (and hence any code actuated by a timer interrupt). The code typically looks like

```
s = spl6();
  . . .  /* mutual exclusion here */
splx(s);
```

where *spl6* locks out the clock and returns the current priority level, and *splx* restores the previous level. If a timer interval expires while the timer interrupt is locked out, the processor delays the interrupt until the priority level is lowered by *splx*. Thus the *splx* routine would be billed for intervals which really belong to its callers. The profiling code could check for this case, examining the stack frame and redirecting the bill appropriately. However, the code for *spl6* and *splx* consists of only 1 or 2 instructions, so it is easy and more efficient to implement them inline, running *sed* over the .s files generated by *cc -S* to replace the subroutine calls with the actual instructions. The timer interrupt is still delayed, but when the interrupt finally is serviced the program counter is pointing directly at the routine which should be charged.

The next trick is to use the VAX-11 stack frame in a clever way. The subroutine calling instruction calls creates a stack frame matching the C declaration

```
struct stack_frame {
        int slot0;              /* initialized to 0 */
        int mask_etc;           /* register save mask and miscellaneous */
        int old_ap;             /* previous argument pointer */
        int old_fp;             /* previous frame pointer */
        int old_pc;             /* return address */
        int reg_sav[n];         /* register save area */
        int nargs;              /* number of arguments */
} *fp;                          /* register fp points at this structure */
```

The first member, *slot0*, is set to zero by the instruction. The DEC manual [1] suggests that this slot be used to indicate the presence of a global error handler, to allow for routine-specific cleanup when the stack is unwound by an exception handler. However, current C code ignores *slot0* entirely. The profiling technique uses this slot to point to the histogram tally cell for the current procedure, as if the stack frame declaration began

```
struct stack_frame {
    int *tally_cell;   /* who to bill when timer goes off */
    int mask_etc;
        . . .
} *fp;
```

An editing script using *sed* changes the procedure entry code produced with *cc -p -S* to achieve the effect of inserting the C code

```
        static int my_tally;
        fp->tally_cell = &my_tally;
```

at the beginning of each procedure. (The compiler could produce this code directly under the -p option. Perhaps it should; but editing the assembly language with *sed* is certainly better for experiments.)

With the above changes, the timer interrupt code becomes

```
Xclockint:
        clkrld();
                /* reprime for next interval */
        if (kernelmode(psl)) {
                if (!is_system_frame(fp)) tickintr++;
                        /* interrupt service */
                else if (fp->slot0)++ *(fp->slot0);
                        /* increment tally cell */
                else if (!is_system_frame(fp->old_fp)) tickintr++;
                else if (fp->old_fp->slot0)++ *(fp->old_fp->slot0);
                        /* caller's tally */
                else tickmiss++;
                        /* no billing address */
        }
        if (--speedup<=0) {
                /* time for normal clock scheduling */
                speedup = SPEEDUP;
                clockint(pc,psl);
        }
```

which can be easily hand coded in assembly language for speed of execution.

The old method of constructing the histogram (for user mode programs) relied on linear transformation of the value of the program counter to select the tally cell. The transformation introduces a spatial quantization error, because portions of two adjacent procedures can be mapped to the same tally cell. The *prof* program must apply a heuristic to apportion the count

between the two procedures. By directly specifying the tally cell for each procedure, the new method reduces the quantization error to nearly zero. If a timer interrupt occurs during the calls instruction, or immediately after entry to a procedure but before the exection of the first instruction within the procedure, then the billing address will have not been set; in such a case billing the caller is appropriate. Timer interrupts which occur during the execution of the assembly-language handler for other device interrupts (such as the disk) may also find no billing address, since the microcode which services interrupt requests does not perform a calls; the frame pointer may even point to the user stack, as when the interrupt occurs while the processor is in user mode. This is also easy to diagnose and handle appropriately. In nearly all cases the new method also reduces the space required for the tally cells, because only *sizeof(int)* bytes suffice for an entire procedure, no matter how large it is.

There is still a time-domain quantization error, but this is unavoidable unless there is a second timer which is independent of the system clock. The speedup factor can be adjusted to give finer resolution, but the overhead of servicing the interrupts will become significant at some point. The timer overrun bit in the control register can be checked to insure that no routine locked out the timer continuously for two consecutive ticks.

4. Adaptations

Any implementation which provides a member such as *slot0* in the stack frame can use a similar technique. (For .s files of compiled C code on the PDP11, an *sed* script can provide the extra slot.) The definition of a system call to provide similar service to code running in user mode is straightforward. [See the following paragraph for an even better idea.] A software interpreter can dynamically change the contents of *slot0* so that the histogram reflects the code being interpreted by the software, rather than the mundane fact that the program counter was always in the interpreter!

To facilitate new uses by code running in user mode, the operating system should provide a profiling service which makes no assumptions about the user's data structure. The system call to invoke the service should specify a subroutine which gets called like a *signal* handler at each timer interrupt. The user would then be free to do anything: profile in the standard manner, profile for a software interpreter, or something unrelated to profiling, such as a special action for each field or frame time of a standard television video signal, or fine control of resource allocation in a prioritized search. Compared with the current scheme in which the operating system knows the data structure, there might be slightly more overhead; but the flexibility would make it worthwhile. It would also considerably simplify the implementation of the current *prof* data structure. There would be no need for the kernel to simulate memory management hardware in order to check user access privileges to the array of tally cells. Particularly in the case of a system with demand paging, a page of tally cells that is "not present" when the timer expires creates a giant headache. Such a generalized profile facility has been implemented on the UNIX/32V system run by Department 1353.

Acknowledgment:

Randall E. Miller assisted the development and implementation of these ideas.

John F Reiser

HO-1353-JFR-jk

John F. Reiser

Atts.
Reference (1)
Appendices 1 & 2
Tables 1,2,3

Reference

1. *VAX-11 Architecture Handbook*, Digital Equipment Corporation, 1979.

Appendix 1: Editor Scripts

Sed script for implementing certain procedures inline:

```
/calls[ ]*$2,_mtpr/c\
\       mtpr 4(sp),(sp)\
\       addl2 $8,sp
/calls[ ]*$1,_mfpr/c\
\       mfpr (sp)+,r0
/calls[ ]*$0,_spl6/c\
\       mfpr $IPL,r0\
\       mtpr $0x18,$IPL
/calls[ ]*$0,_spl0/c\
\       mfpr $IPL,r0\
\       mtpr $0,$IPL
/calls[ ]*$1,_splx/c\
\       mfpr $IPL,r0\
\       mtpr (sp)+,$IPL
```

Here 'IPL' has been used for clarity in place of '0x12'.

The *sed* script to set tally cell address and perform subroutine entrance counts directly changes

```
_main:
        .word   .R1
        jbr     L13
L14:
        movab   L15,r0
        jsb     mcount
        .data
        .align  2
L15:    .long   0
        .text
```

into

```
_main:
        .word   .R1
        .lcomm  L15a,8
        moval   L15a+4,r0
        movl    r0,(fp)
        sobgeq  -(r0),L15
        jsb     mcount
L15:
        jbr     L13
L14:
```

The script itself is:

```
/\.word \.R[0-9]*/{
        n
        /jbr[    ]*L[0-9]*/{
                h
                N
                s/.*\n//
                /L[0-9]*:/{
                        H
                        N
                        s/.*\n//
                        /movab[ ]*\(L[0-9]*\),r0/{
                                s//.lcomm       \1a,8/
                                p
                                s/[^L]*\(L[0-9]*\).*/    moval  \1a+4,r0/
                                p
                                i\
                                \ movl  r0,(fp)
                                s//      sobgeq  -(r0),\1/
                                p
                                i\
                                \ jsb    mcount
                                s//\1:/
                                n
                                /jsb[    ]*mcount/x
                                p
                        :flush
                                s/.*//
                                N
                                /\.text/d
                                bflush
                        }
                }
                x
                p
                g
        }
}
```

(Note that *mcount* also requires adjustment.)

Sed script to add slot in stack frame for .s compiled C code on the PDP11:

```
/-[0-7]*(r5)/s/\(-[0-7]*\)(r5)/\1-2(r5)/g
/mov   r5,/{
        n
        s/\(add \$-[0-7]*\)/\1-2/
}
```

The new slot is at -8(r5), between the saved registers and the locals. To actually allocate the space, add

```
clr -(sp)
```

just before the

```
jsr pc,(r0)
```

in *csv*, or just before the

```
tst -(sp)
```

in *csav*. Note that certain debugging programs might get confused about where the local variables reside after this change.

Appendix 2: Tuning UNIX/32V

The UNIX/32V system on the VAX-11 run by Department 1353 includes the new profiling scheme as a matter of course. The cost is approximately 14K bytes of memory and a negligable percentage of system cpu time. Analysis of the profiles revealed an efficiency bug (since fixed), another efficiency bug in UNIX itself (even on the PDP11; fixed on the VAX-11), suggested that several routines be implemented inline, and continues to increase understanding of the dynamic behavior of the system. The routines *mtpr, mfpr, spl6, spl5, spl0, splx, bcopy, copyin, fubyte, fuword* were implemented inline with *sed* script. Tables 1, 2, and 3 contain the initial portions of a profile of slightly more than 4 hours of operation during the middle of the day on April 10, 1980. A *tick* is 1/300 second, *Scbbase* represents interrupt service performed in assembly language, and *_u* represents missed ticks.

routine	calls	ticks	usec/call
_idle	952376	2214985	7752
_trap	1213092	39299	108
_getblk	655803	36968	188
Scbbase	0	36334	121101222
_wakeup	537313	29077	180
_namei	25123	23612	3133
_rdwr	613378	22920	125
_sleep	474788	21327	150
_readi	456516	18986	139
_swtch	861800	18032	70
_ttread	333072	17280	173
_getc	1799000	17198	32
_copyout	198212	16738	281
_hpstart	549845	16334	99
_hpintr	398464	15905	133
_canon	618906	15853	85
_iget	66671	14994	750
_hpstrat	254986	14239	186
_dzkrint	380465	12262	107
_resume	285636	11993	140
_putc	1124022	11572	34
_ttyinpu	381200	11535	101
_brelse	693987	10329	50
_hpustar	321180	10012	104
_bmap	289174	8688	100
_kmcload	119253	8499	238
_b_to_q	440401	8174	62
_bread	486207	8029	55
_kmcrint	119964	7764	216
_mbastar	254985	7385	97
_memall	35396	7350	692
_setjmp	808001	7047	29
_iomove	348729	7043	67
_notavai	693960	6575	32
_qswtch	384993	6525	56
_incore	237184	6487	91
_ttwrite	130500	6461	165
_breada	121552	6217	170
_passc	379011	5438	48
_clearse	75957	5355	235
_dzkstar	191687	5178	90
_writei	165979	5159	104
_dzkread	332923	5134	51
_kmcxint	118011	4528	128
_read	451444	4449	33
_dzkkint	118590	4269	120
_setrq	861803	4043	16
_memfree	30002	3936	437
_swapin	6142	3350	1818
_sched	1	3282	10938906

Table 1. Profile of UNIX/32V by decreasing ticks

routine	calls	ticks	usec/call
_getc	1799000	17198	32
_trap	1213092	39299	108
_putc	1124022	11572	34
_idle	952376	2214985	7752
_clock	925562	0	0
_setrq	861803	4043	16
_swtch	861800	18032	70
_setjmp	808001	7047	29
_brelse	693987	10329	50
_notavai	693960	6575	32
_getf	677113	2637	13
_issig	664933	877	4
_getblk	655803	36968	188
_canon	618906	15853	85
_rdwr	613378	22920	125
_hpstart	549845	16334	99
_wakeup	537313	29077	180
_setpri	503854	2835	19
_bread	486207	8029	55
_setrun	480300	81	1
_sleep	474788	21327	150
_readi	456516	18986	139
_read	451444	4449	33
_b_to_q	440401	8174	62
_aston	427626	0	0
_min	413568	1180	10
_hpintr	398464	15905	133
_qswtch	384993	6525	56
_ttyinpu	381200	11535	101
_dzkrint	380465	12262	107
_subyte	379011	2332	21
_passc	379011	5438	48
_iomove	348729	7043	67
_ttread	333072	17280	173
_dzkread	332923	5134	51
_hpustar	321180	10012	104
_bmap	289174	8688	100
_resume	285636	11993	140
_uchar	277003	1596	19
_prele	274662	963	12
_hpstrat	254986	14239	186
_mbastar	254985	7385	97
_iodone	254984	3174	41
_incore	237184	6487	91
_copyout	198212	16738	281
_dzkstar	191687	5178	90
_writei	165979	5159	104
_disksor	163962	2137	43
_write	161924	1683	35
_ttstart	156173	2297	49

Table 2. Profile of UNIX/32V by decreasing calls

routine	calls	ticks	usec/call
Scbbase	0	36334	121101222
_sched	1	3282	10938906
_u	0	182	606606
_kmcwrit	2	190	316635
_stop	2	5	8333
_idle	952376	2214985	7752
_putchar	120	198	5499
_update	513	759	4931
_binit	1	1	3333
_namei	25123	23612	3133
_exit	1957	1345	2291
_smount	3	2	2222
_procdup	1998	1262	2105
_newproc	1998	1109	1850
_swapin	6142	3350	1818
_exece	4199	2283	1812
_tloop	317	170	1787
_fork	1997	780	1302
_wait	2040	775	1266
_kill	4654	1419	1016
_printf	4	1	833
_xswap	3430	806	783
_iget	66671	14994	750
_xalloc	1902	415	727
_memall	35396	7350	692
_getxfil	1938	398	684
_signal	82	15	610
_ptexpan	244	40	546
_chgprot	3991	604	504
_memfree	30002	3936	437
_expand	13127	1674	425
_ialloc	1932	240	414
_setregs	1901	223	391
_flushtt	558	61	364
_copyseg	19619	2140	364
_consope	33	3	303
_clrbuf	22406	1939	288
_copyout	198212	16738	281
_chdirec	353	26	245
_kmcload	119253	8499	238
_clearse	75957	5355	235
_ptrace	57	4	234
_link	352	24	227
_writep	2551	167	218
_kmcrint	119964	7764	216
_iexpand	28731	1864	216
_falloc	8357	532	212
_itrunc	2199	130	197
_getblk	655803	36968	188
_hpstrat	254986	14239	186

Table 3. Profile of UNIX/32V by decreasing time per call