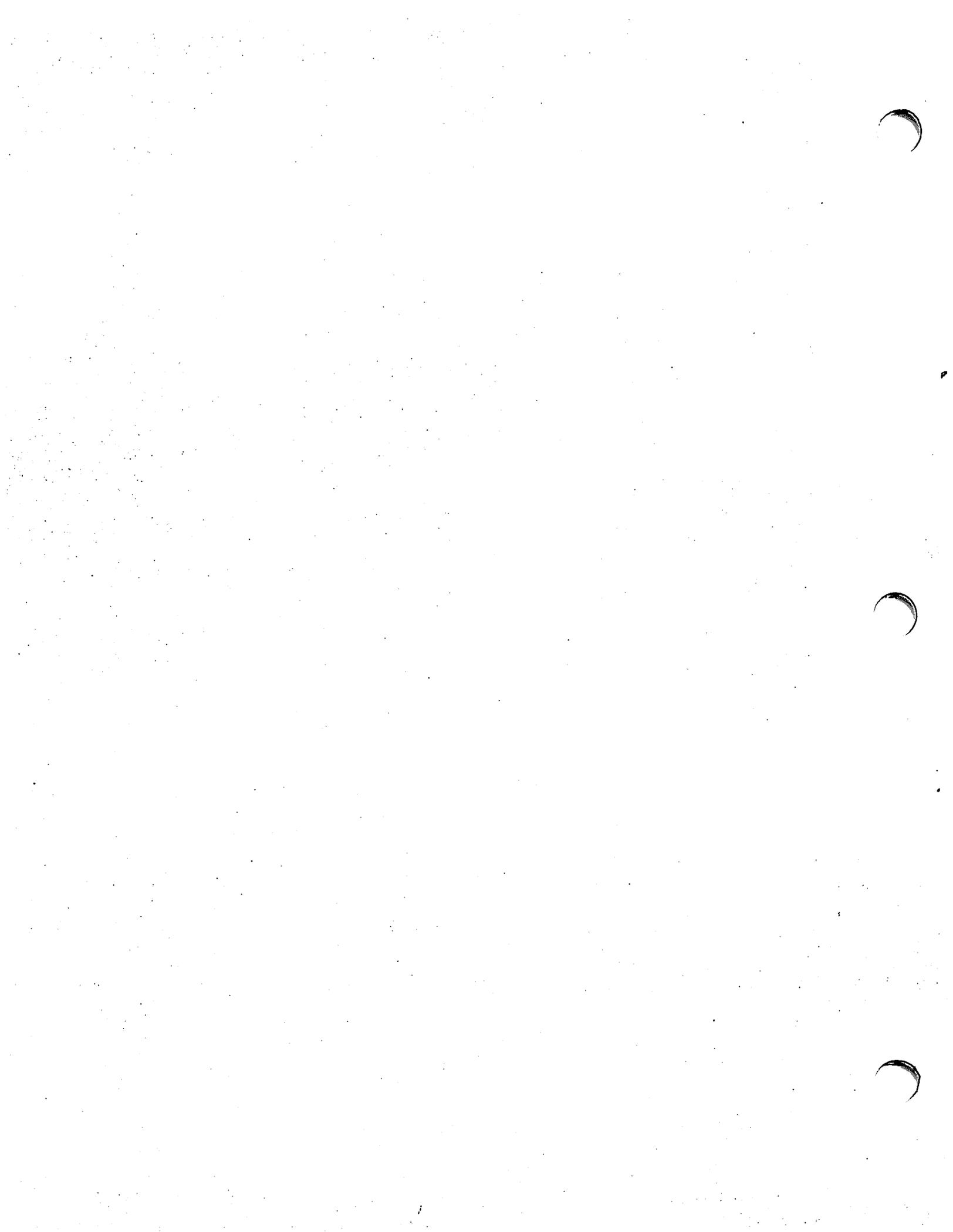ANDERSON COLLEGE

Function: This program summarizes a range of log files into one
new file compatible with program LOGRPT. The log files analyzed
are purged by this program.

Program documentation: For detail information on files, messages,
and errors, see source listing lines 3/56.

Run instructions:

1. This program may be run either as job or session.

2. User must have system manager capability.

3. :RUN LOGANA.PUB.SYS

4. FILE NOT FOUND
    On beginning execution the program purges SORTLOG. This
    message appears if SORTLOG does not exist. No response
    is required.

5. ENTER BEGINNING LOG NBR? XXXX
    Enter 4 digit number of first log file in range. Example:
    if LOG∅123 is the first in the range, enter ∅123. (CR not
    required.)

6. ENTER ENDING LOG NBR? XXXX
    Enter 4 digit number of last log file in range.
    NOTE: The last log number must be greater than the first
           and all log files in the range must exist. If a
           log file within the range is missing, a dummy may
           be created with the command :BUILD LOGXXXX;REC=-5∅8,
           1,V,BINARY;DISC=1,1,1.

7. The log files within the range are now summarized into
    SORTLOG and purged. A maximum of 20 jobs may be open at
    one time in the log files. Job records are not carried
    over between log files. The summary records in SORTLOG
    are now arranged in chronological order and contain the
    following:

              A. Time stamp-on
              B. Job type and number
              C. User name
              D. Job name
              E. Group name
              F. CPU time (seconds)
              G. Connect time (minutes)
              H. Records processed (all files except printer
                  (LDEV=6) and terminals (LDEV>1∅)).
              I. Lines printed (records processed LDEV=6).
              J. Time stamp-off

8. <u>END OF PROGRAM</u>

9. To prepare the file SORTLOG for use by program LOGRPT,
   SORT must be run as follows:

   ```
   :FILE INPUT=SORTLOG
   :FILE OUTPUT=SORTLOG
   :RUN SORT;STACK=10000
   >KEY AT 16 IS 8 LONG
   >KEY AT 32 IS 8 LONG
   >KEY AT  8 IS 8 LONG
   >KEY AT  0 IS 6 LONG
   >:EOD
   END OF PROGRAM
   ```

Anderson College
Computing Center

Jan. 1974

PROGRAM | LOGRPT | sourcefile=LOGRSRC

Function: This program produces a report of usage for all jobs,

users, groups, and accounts. Input is from the file SORTLOG as

prepared by programs LOGANA and SORT plus a rate file RATES. The

output is a printed report which shows the following information

for each job:

    A. Job name
    B. Job number
    C. Date-on
    D. Time-on
    E. Time-off
    G. Connect time and connect time charge
    H. CPU time and CPU time charge
    I. Records processed and record processing charge
    J. Lines printed and printer charge
    K. Total job charge.

The output is ordered by user, group, and account and totals are

given at each level plus grand totals.

Program Documentation: For detail information on files (including

file RATES), messages, and errors, see source listing lines 3/55,

79/95. Note: This program requires that procedure DOLLAROUT be

in SL.PUB.SYS as an intrinsic.

Run instructions:

    1. This program may be run either as job or session.
    2. User must have system manager capability.
    3. :RUN LOGRPT
        The report is now printed on the line printer.
    4. END OF PROGRAM

The file SORTLOG remains intact and LOGRPT may be rerun.

                                        Anderson College
                                        Computing Center

                                        Jan. 1974

Program | FOOTBALL | Basic program file = FOOTBALL
                                       Basic data file   = DATA1
                                       Basic data file   = DATA2

Function: This program simulates a regulation football game
using two terminals. Each team's plays (offensive and defensive)
are called by a "quarterback" at a separate terminal. The files
DATA1 and DATA2 contain probabilities for each kind of play
(DATA1) and defensive skew factor (DATA2).

Run Instructions:

1. The I/O table (MPE) must be modified to allow terminals
    to accept data (JAID).

2. Determine the LDEV # for the "visitors'" terminal. This
    may be done by logging on and running WHO.PUB.SYS.

3. Enter a :DATA command* on the "visitors'" terminal.

4. Log on* the "home" terminal and determine its LDEV #.

5. Enter these file commands on the "home" terminal:
    :FILE TEAM1;DEV=("home" terminal LDEV #)
    :FILE TEAM2;DEV=("visitors'"terminal LDEV #)

6. Run the program as follows:
    :BASIC
    >RUN FOOTBALL
         •
         •
         •

7. The "visitors'" terminal will be released when

    a. FOOTBALL ends, or
    b. The home terminal hangs up, or
    c. The visitors' terminal enters :EOD.

8. All rules, options, etc are explained as the program runs.

*   The DATA command in Step 3 must use exactly the same job
    identification as the user in Step 4.

Example

Home Terminal

Visitors' Terminal

```
:HELLO PRO.GAMES
:FILE TEAM1;DEV=14
:FILE TEAM2;DEV=15
:BASIC
>RUN FOOTBALL
        .
        .
        .
```

```
:DATA PRO.GAMES
```

```
        .
        .
        .
```

Anderson College
Computing Center
Jan. 1974

PROGRAM WHO   sourcefile = WHOSRC

Function: This program, when executed, prints a

formatted dump of the information available from

the intrinsic WHO. The output is to $STDLIST; a

sample is shown below.

```
:RUN WHO.PUB.SYS

YOUR CURRENT ATTRIBUTES ARE AS FOLLOWS:
USER = FIELD
HOME GRP = PUB
LOGON GRP = USERS
ACCT = SUPPORT
INTERACTIVE, DUPLICATIVE, SESSION

USER ATTRIBUTES: GL, AL, AM,
FILE ACCESS:  SF, ND
CAPABILITY CLASS: IA, BA
LOGICAL DEV # = 13
```

END OF PROGRAM

Anderson College
Computing Center

Jan. 1974

PROCEDURE | DOLLARIN(word,string,length); |

VALUE length;
INTEGER length;
DOUBLE word;
BYTE ARRAY string;   sourcefile=SLPUBSRC

Function: This procedure converts an ASCII string contain-
ing a dollar amount into a double word representing the
corresponding number of cents. Dollar signs ($), commas (,),
blanks ( ), and plus signs (+) in the ASCII string are ignored.
A minus sign (-) or the letters (CR) anywhere in the string will
cause the returned value to be negative. Any other non-numeric
characters except the decimal point (.) will cause an error
condition.

    word   The double word into which the converted value
            is placed.

    string The byte array containing the ASCII string to
            be converted.

    length The length of the byte array (cannot be nega-
            tive).

The DOLLARIN command can result in the following condition
codes:

    CCE   Successful conversion.

    CCG   Illegal character found in string.

    CCL   (This condition code is not returned.)

Anderson College
Computing Center

Jan. 1974

PROCEDURE | DOLLAROUT(word,string,length,type);

VALUE word,length,type;
DOUBLE word;
INTEGER length,type;
BYTE ARRAY string;   sourcefile=SLPUBSRC

Function: This procedure converts a double word into an ASCII
string with the two low order digits to the right of the decimal
point. A dollar sign ($) and/or commas are optionally included in
the string. The double word is assumed to represent an integer
number of cents, not dollars.

    word   The double word to be converted to ASCII code.
               Its value is interpreted as an integer number of
               cents, and may be positive or negative.

    string   The byte array into which the converted value is
               placed. The result is right justified.

    length   The length of the byte array. (Note that 4≤length
               ≤132.)

    type   An integer identifying the kind of conversion de-
               sired (see <u>HP 3000 Compiler Library</u>: INEXT'):

               type<0   -2 PFw.2 format,example: 1234.56
               type=0   -2 PNw.2 format,example: 1,234.56
               type>0   -2 PMw.2 format,example: $1,234.56

The DOLLAROUT command can result in the following condition codes:

    CCE   Successful conversion.

    CCG   No conversion; length>132.

    CCL   No conversion; length not long enough to contain
               the results.

PROCEDURE   DUMPSTACK;   sourcefile = SLPUBSRC

Function: This procedure, when called, prints the following

information: the contents of DL, DB, Q, S, Z, Index, Status

and P followed by a dump of the stack from DB+$\emptyset$ to S.   The

values printed correspond to the values just before the call

to DUMPSTACK is made.   All values are in octal and are

clearly labeled.   Output is to file "LIST" which defaults to

device "LP" with characteristics ASCII, CCTL, WRITE ONLY.

The output file must have a record width of 66 words or

greater.

With this procedure the condition code remains unchanged.

Any errors encountered will cause the program to print

"*ERROR" followed by termination.

Anderson College
Computing Center

Jan. 1974

PROCEDURE DUMPREG;   sourcefile = SLPUBSRC

Function: This procedure prints the current contents

of the stack registers (DB, DL, Z, STAT, X, Q, S) on

$STDLIST when called. The values printed correspond

to the values just before the call to DUMPREG is made.

Note that DL, Z, Q, and S are DB relative and that DL

is normally negative in two's complement form. This

is useful in debugging programs where stack overflow

or underflow occurs.

With this procedure the condition code remains unchanged.

Sample output:

```
    DB = %156250
    DL = %177732
     Z = %001605
    ST = %060151
     X = %0000007
     Q = %000273
     S = %000314
```

PROCEDURE | READCARD (buffer);

ARRAY buffer;   sourcefile = SLPUBSRC

Function: When called, this procedure causes a card to be read
from the card reader (logical device #5), and stores the data
in column-binary form. Each card column is stored in the cor-
responding word of buffer with the bit in row 12 stored in
bit 11 and the bit from row 9 stored in bit 0.* READCARD
bypasses the MPE file system. Therefore the following should
be noted:

1. A :DATA card is not required to precede the data.

2. Cards with a colon in column 1 are not intercepted
   or otherwise recognized by MPE. Therefore these
   cards may be read as data.

3. No check is made for other users on the card reader.
   Care must be used not to "steal" another user's
   data.

4. An :EOF card is not needed or required. The user
   must determine programatically when the last card
   has been read.

buffer   The logical array into which the column binary data
          is stored. It must be at least 80 words in length.

The READCARD command can result in the following condition codes:

   CCE   The card read was successful.

   CCL   An error occurred during the read operation or insuf-
         ficient stack space exists for the operation.

   CCG   (This condition is not returned.)

This procedure must be compiled by a user with privileged mode
capability.

* See 30106-90001 Maintenance Manual: 30106A Card Reader
  Subsystem for details.

                                        Anderson College
                                        Computing Center
                                        Jan. 1974

PROCEDURE   HOLTH (source,dest,cnt,code);

ARRAY source;
INTEGER cnt;
LOGICAL code;
BYTE ARRAY dest;   sourcefile = SLPUBSRC

Function: This procedure converts data from column binary
form to ASCII or IBM 1620 BCD form. It is normally used with
procedure READCARD to process cards containing part ASCII and
part binary data.

    source   The word array containing the column binary data
               to be converted.

    dest   The byte array into which the converted output
               is stored.

    cnt   An integer whose value indicates the number of
               words to be converted.

    code   A logical word indicating the type of conversion:

                    TRUE = IBM 1620 BCD,
                    FALSE = ASCII.

With this procedure the condition code remains unchanged.

Running Compilers as programs instead of subsystems

Occasionally it is useful to have more than one version of a compiler available at one time. This may be done by storing one version and running it as a program.

Example: a new version of SPL is received but the old version is kept until the new one is checked out. The following commands save the old version and put up the new one:

```
:HELLO MANAGER.SYS
:RENAME SPL,SPLOLD
:FILE TAPE;DEV=TAPE
:RESTORE *TAPE;SPL;SHOW      (new SPL is up as
                                 subsystem)
:BYE
```

The new version may now be invoked with the :SPL command in the usual manner. To call the old version it is first necessary to put in file equations using the formal designators of the compiler. Example:

```
:FILE SPLTEXT=MYPGMSR
:FILE SPLUSL=$NEWPASS
:FILE SPLLIST;DEV=LP
:FILE SPLMAST=$NULL
:FILE SPLNEW=$NULL
```

To run the compiler, give a RUN command with parameter=31. Example:

```
:RUN SPLOLD.PUB.SYS;PARM=31
```

With appropriate file names, this procedure should work for other compilers as well.

<div align="right">

Anderson College
Computing Center

</div>

ANDERSON COLLEGE COMPUTING CENTER

Report on Multiprogramming Efficiency
and File System Overhead Under MPE B.1.S4

November 8, 1973

Introduction

The accounting system of MPE version B does not count CPU time used by the system on behalf of a user (such as I/O handling) as part of the user's CPU time. This experiment was conducted to determine:

1. If the unaccounted for CPU time could be correlated with the number of records processed;

2. What the average overhead is per record processed;

3. What additional overhead is incurred by multiprogramming.

Conclusions

1. Unaccounted for CPU time correlates very highly with the number of records processed.

2. The average overhead per record processed is approximately 65 msec.

3. The additional overhead incurred by multiprogramming is about two percent when code can be shared.

Description

A single job was selected which made heavy use of the file system, was short enough to be tested in a reasonable time, and which could be run simultaneously by several users. The job was to compile and prepare an SPL program of 402 lines of source code (LOGRPT). Paper tapes were prepared with the following commands:

```
:HELLO CR.CC
:SPLPREP LOGRSRC
:BYE
```

The source file (LOGRSRC) included a NOLIST command in the second line.

Initially this job was run, from paper tape, with no other users on the system. Following this session, the same job was run from three

terminals with all sessions starting within a 20 second period. In
each case the session was run entirely from paper tape, eliminating
human response-time as a factor. The following table shows the results
obtained. All times are in seconds; sessions 2-4 were run at the same
time.

Job	Connect time	CPU time	Records processed
#S1	107.9	27	1344
#S2	329.6	28	1267
#S3	297.5	27	1232
#S4	283.8	25	1155

Discussion

1. The formula CON=CPU+T*REC was applied to the above data where:

   > CON is total elapsed time for one or more sessions,
   > CPU is total CPU time for one or more sessions,
   > T is a factor to be assigned as overhead time per record,
   > REC is the number of disc records processed by one or
   >   more sessions.

   In the case of #S1 the factor T was found to be 60.1 msec. per
   record while in the combined run of #S2,3,4 it was found to be
   67.6 msec. The difference is probably due to the additional
   system overhead involved in running three simultaneous jobs.

   The close agreement between these figures would indicate that a
   value of 60-70 msec. is very consistent when averaged over a
   medium sized job.

2. The total elapsed time for #S2,3,4 was 329.6 seconds. This
   compares closely with 323.7 seconds which is triple the time
   for #S1. The difference is less than 2%. Since sessions #S2,3,4
   were run very nearly in synchronism, they would be expected to
   be either I/O bound or compute bound simultaneously, thus negat-
   ing any gain in efficiency due to multiprogramming. In actual
   experience the sessions seemed to leap-frog past each other and
   #S2 which started first was the last to finish. This is probably
   due to the nature of code sharing as noted below.

3. The following table shows the files closed by each session with
   the number of records processed.

	File	Records	#S1	Session		
				#S2	#S3	#S4
1.	LOAD.PUB.SYS	2	X	X		
2.	SL.PUB.SYS	47	X	X		
3.	SPL.PUB.SYS	63	X	X		
4.	SPL.PUB.SYS	2	X	X	X	X
5.	$NEWPASS	384	X	X	X	X
6.	TEMPLIST	Ø	X	X	X	X
7.	TEMPCODE	282	X	X	X	X
8.	SPLINTR.PUB.SYS	20	X	X	X	X
9.	LOGRSRC	422	X	X	X	X
10.	SPLLIST	8*	X	X	X	X
11.	LOAD.PUB.SYS	2	X		X	
12.	SL.PUB.SYS	52	X		X	
13.	SEGPROC.PUB.SYS	23	X		X	
14.	SEGPROC.PUB.SYS	2	X	X	X	X
15.	$NEWPASS	37	X	X	X	X
16.	$OLDPASS	6	X	X	X	X

    * $STDLIST, not counted in totals.

It can be seen that #S2 bore the cost of allocating the compiler (files 1-3) while #S3 bore the cost of allocating the segmenter (files 11-13). Thus in each case the other two sessions were given a "free ride" because they could share the allocated code. The efficiency thus gained appears to nearly (but not entirely) compensate for the added overhead involved in multiprogramming.

HP 3000

2610A Line Printer 200LPM

2893A Card Reader 600CPM

2888A disc 47M bytes

7970B Mag Tape 800BPI

2895B Paper Tape Punch 75CPS

Multiplexer Channel

Core Module 32K words

IOP

Core Module 32K words

CPU

2762A Console Printer

Terminal Multiplexer

SPL
FORTRAN
BASIC
COBOL
SORT
EDIT
STAR

Vadic Modems

20 Terminals

ANDERSON COLLEGE

HP 2000E

7900A disc 4.8M bytes

7900A disc 4.8M bytes

7970B Mag Tape 800BPI

Terminal Multiplexer

2100A 16K words

BASIC

68