1114

Bell Laboratories Cover Sheet for Technical Memorandum

Title- Synthetic Process for UNIX Date- September 24, 1976

TM- 76-8234-17
76-9156-2

Other Keywords- Performance Evaluation
Workload Measurement
Interprocess Communication

Author	Location	Extension	Charging Case- 70107-003
D. R. Bernstein	MH 2F-245	2008	Filing Case- 40952-1

ABSTRACT

A synthetic job performs a parameter-specified amount of processor cycles and disk I/O operations. Such jobs have been used successfully in measurement experiments. Patterned after widely publicized versions written in Fortran and PL/1, a UNIX version has been implemented in the C language. Input/output options for this version comprise read, write, getc, putc, getw, putw, as well as messages and pipes. The synthetic job concept has been extended further by providing facilities for issuing an arbitrary sequence of system calls such as fork, exec, kill, nice, sleep and wait. With these facilities, networks of cooperating synthetic processes can be constructed as models of applications. The synthetic process writes self-timing information into a report file. Some measurements of system calls comparing different hardware (PDP-11/45 and /70) and software (UNIX and MERT) are presented for illustration.

Pages Text 6 Other 14 Total 20

No. Figures 1 No. Tables 1 No. Refs. 4

E-1932-U (6-73)SEE REVERSE SIDE FOR DISTRIBUTION LIST

DATE FILE

BELL TELEPHONE LABORATORIES, INC.

DISTRIBUTION
(REFER GEI 13.9-3)

TM-76-8234-17
TM-76-9156-2

COMPLETE MEMORANDUM TO

CORRESPONDENCE FILES

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

DATE FILE COPY
(FORM E-1328)

REFERENCE COPIES

ARTHURS, E
*AVERILL,R M,JR
BALDWIN,GEORGE L
BRANDT,RICHARD B
*CANADAY,RUDD H
COHEN,AARON S
COPP,DAVID H
*DOLOTTA,T A
DOWD,P G
FARRELL,JAMES WILLIAM
FEDER,J
NENS,J A
HI,R C
,ANDREW D,JR
*IRVINE,M M
*ISENMAN,M E
JENSEN,PAUL D
*JOHNSON,W C
*KAPLAN,MICHAEL M
<KAYEL,R G
>KLEIN,RUTH L
LONDON,H S
LONG,P F
LUDERER,GOTTFRIED W R
MARANZANO,J F
<MUENZER,T B
PEREZ,CATHERINE D
*PRICE,A C,JR
RALEIGH,T M
RIDGWAY,WILLIAM C,III
RITACCO,J E
ROBINSON,H E
<STEIGERWALT,R A
*STUCK,B W
SWIFT,R E
TAGUE,BERKLEY A
THAYER,P H,JR
VOGEL,GERALD C
*WOLFE,ROBERT R
*YOSTPILLE,J J
40 NAMES

COVER SHEET ONLY TO

CORRESPONDENCE FILES

COVER SHEET ONLY TO

4 COPIES PLUS ONE
COPY FOR EACH FILING
CASE

ACKERMAN,A FRANK
AHO,ALFRED V
AHRENS,RAINER B
ALBERTS,BARBARA A
ALCALAY,D
ALLEN,JAMES R
ALLISON,C E,JR
ALMQUIST,R P
AMITAY,N
AMOSS,JOHN J
ANDERSON,C R
ANDERSON,KATHRYN J
ANDERSON,L S
ARMSTRONG,D B
ARNDT,DENNIS L
ARNOLD,GEORGE W
ARNOLD,S L
ARNOLD,THOMAS F
ARTIS,H P
ATAL,BISHNU S
AVIL,H J
BACCASH,JEANNE M
BACKUS,C F,SR
BAKER,BRENDA S
BALLARD,E D,JR
BARRESE,A L
BASEIL,RICHARD J
BATTAGLIA,FRANCES
BAUER,HELEN A
<BAUER,WOLFGANG F
BAUGH,C R
BAYER,D L
BECKER,RICHARD A
BECKETT,J T
BECK,R P
BEL BRUNO,KATHLEEN A
BENNETT,JOHN EDWARD
BENNETT,WILLIAM C
BERGLAND,G D
BERNSTEIN,DANIELLE R
BERNSTEIN,L
BERRYMAN,R D
BERTH,R P
BEYER,JEAN-DAVID
BIANCHI,M H
BICKFORD,NEIL B
BILOWOS,R M
BIRCHALL,R H
BIREN,IRMA B
BLAZIER,S D
BLINN,J C
BLUE,JAMES L
BLUM,MARION
BODEN,F J
BOLSKY,MORRIS I
BONANNI,L E

COVER SHEET ONLY TO

BOURNE,STEPHEN R
BOWEN,F W
BOWERS,JERRY L
BOWLES,J B
BOWRA,JAMES W
BOWYER,L RAY
BOYCE,W M
BOYLE,GERALD C
BRAND,JOE E
BRANTLEY,JOAN T
BROEK,H W
BROOKS,CATHERINE A
BROWN,C W
BROWN,JAMES W
BROWN,W R
BROWN,W STANLEY
BUCHANAN,D N E
<BUCZNY,F A,JR
BULFER,A F
BURES,C
BURGESS,JOHN T,JR
BURNETTE,W A
BURROWS,THOMAS A
BUSCH,KENNETH J
BYRNE,EDWARD R
CALLAHAN,R L
CANNON,T B
CARDOZA,WAYNE M
CARRAN,JOHN H
CARR,DAVID C
CASPERS,BARBARA E
CASTELLANO,MARY ANN
CATO,H E
CAVINESS,JOHN D
CERMAK,I A
CHAFFEE,N F
CHAI,D T
CHAMBERS,B C
CHAMBERS,J M
CHANG,HERBERT Y
CHAPPELL,S G
CHEE,T
CHEN,STEPHEN
CHEN,T L
CHERRY,LORINDA L
CHIANG,T C
CHODROW,M M
CHRIST,C W,JR
CIESLAK,THOMAS J
CLAYTON,D P
CLOUTIER,J
CLYMER,J C
COBEN,ROBERT M
COHEN,HARVEY
COLE,LOUIS M
COLE,MARILYN O
COLLINS,J P
COMMRADE,GERALDINE
CONNERS,RONALD R
COOK,T J

COVER SHEET ONLY TO

COOPER,ARTHUR E
COREY,D A
COSTANTINO,BASIL B
COSTELLO,PETER E
COSTON,W P
CRACOVIA,E V
CRAGUN,DONALD W
CRANE,R P
CROWE,MARGARET M
CRUME,L L
CSASZAR,MARYANN
CUNNINGHAM,S J,JR
DAVIDSON,CHARLES LEWIS
DAVIS,D R
DAVIS,R DREW
DE GRAAF,D A
DE JAGER,D S
DE PAOLA,THELMA T
DENISON,JOANNA
DENTON,R T
DERMOND,FAITH L
DESENDORF,JUDITH L
DESMOND,JOHN PATRICK
DEUTSCH,DAVID N
DEVLIN,N V
DEVLIN,SUSAN J
DI GIACOMO,J G
DI PIETRO,R S
DICKMAN,BERNARD N
DIMMICK,JAMES O
DOBLMAIER,A H
DOMBROWSKI,F J
DONKIN,A E
DONNELLY,MARGARET M
DONOFRIO,LOUIS,J
DOWDEN,DOUGLAS C
DRAKE,LILLIAN
DRENNING,C R
DRUMMOND,R E
D'ANDREA,LOUISE A
DSTEFAN,D J
DUDICK,ANTHONY L
DUDLEY,ELIZABETH H
DUFFY,F P
DUNN,J C
DWYER,T J
EDDY,MICHAEL R
EDELSON,DAVID
EDMUNDS,T W
EITELBACH,DAVID L
EL-SHABAZZ,A M
ELY,T C
EPLEY,ROBERT V
ERRICHIELLO,PHILIP M
ESSERMAN,ALAN R
ESTES,VIRGINIA DANIELLE
ESTOCK,RICHARD G
ESTVANDER,R A
EVENSON,E K
EVERETT,W W

COVER SHEET ONLY TO

FABISCH,M P
FACTOR,ROBERT M
FASCIANO,V
FAULKNER,ROGER A
FEINBERG,HENRY R
FELS,ALLEN M
FELTON,WILLIAM A
FEMPER,NANCY L
FISCHER,HERBERT B
FLANDRENA,R
FLEISCHER,H I
FLYNN,MARY L
FOLEY,G
FONG,K T
FONTENOT,SHARON M
FOO,YEOW PIN
FORTNEY,V J
FOUGHT,B T
FOUNTOUKIDIS,A
FOWLER,BRUCE R
FOWLER,C F
FOX,PHYLLIS A
FOY,J C
FRANKLIN,DANIEL L
FRANKS,RICHARD L
FRANK,AMALIE J
FRANK,H GREGORY
FRANK,RUDOLPH J
FRANZ,ANN M
FRASER,A G
FREEDMAN,MARVIN I
FREEMAN,K G
FREEMAN,R DON
FROST,H BONNELL
FULTON,A W
GAJEWSKA,HANNA
GANNON,T F
GARCIA,R F
GATES,G W
GAY,FRANCIS A
GEARY,M J
GELBER,CHERON L
GEPNER,JAMES R
GERGOWITZ,E B
GEYLING,F T
GIBB,KENNETH R
GIBSON,H T,JR
GILLON,ALEX C
GILL,R J
GIMPEL,J F
GINSBERG,N
GLASER,W A
GLASSER,ALAN L
GLUCK,F G
GODWIN,R E
GOETZ,FRANK M
GOGUEN,N H
GOLABEK,RUTH T
GOLDSMITH,LAWRENCE D,JR
GOLDSTEIN,A JAY
...
749 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)

F ORY SPECIFICATION..

LETE MEMO TO:
8234-AMTS 823-DPH 82-DIR

COVER SHEET TO:
COCSPM = COMPUTING SYSTEM PERFORMANCE MEASUREMENT, SIMULATION, ACCOUNTING
COPRDB = COMPUTER PROGRAM DEBUGGING, MEASUREMENT AND TESTING
UNOS# = UNIX/OPERATING SYSTEM
UNSU# = UNIX/SERVICE,UTILITY PROGRAMS

HO CORRESPONDENCE FILES
HO 5C101

TM-76-8234-17
TOTAL PAGES 20

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.

Bell Laboratories

Subject: Synthetic Process for UNIX
Case- 70107-003 -- File- 40952-1

date: September 24, 1976

from: D. R. Bernstein

TM: 76-8234-17
76-9156-2

MEMORANDUM FOR FILE

Introduction

A synthetic job is a program that simulates a load (in terms of units of work) imposed on a system. It consumes specified amounts of resources (I/O, CPU and memory). It can be used in place of real jobs in a traditional benchmark (a representative subset of a computer load) to compare the relative performance of different hardware or software features of a computer system.

The motivation for pursuing the use of a synthetic job instead of benchmarking is well documented by J.F. Maranzano (Ref. 1). Basically, a synthetic job does not need a specific data base that must be moved from system to system. It has parameters that specify what resources are to be consumed. It is more portable and flexible than a set of benchmark jobs.

The synthetic job can be used with any measuring tool desired by the analyst such as a hardware monitor, software monitor or programmable clock. It can also do its own (rather gross) measuring by timing itself.

A process in the UNIX environment is logically equivalent to a job in the batch world. The synthetic job that is to be described is similar in purpose to the synthetic job available in Fortran written by R.L. Klein and J.E. Ritacco (Ref. 2). The present synthetic process is written in C and is intended to be used on a Unix or Mert operating system. This tool can be used to compare different systems by running experiments which use identical resources. It also is useful in modeling (at a gross level of detail) existing or new dedicated applications which use a network of cooperating and communicating processes.

It allows the user to specify:

1. Sequences of system calls to fork and exec to new synthetic processes.

2. Communication via pipes or messages between cooperating synthetic processes.

It can give insight into questions such as:

How does the addition of a new process affect the performance of the process already running?
Should piping or messages be used if a given number of bytes needs to be transmitted?

Overview

The synthetic process (SP) consists of two parts:

A. The monitor part of the program which:

1. Reads user input

2. Initializes the necessary parameters

3. Starts the passes through the resource usage

4. Does timing before and after that resource usage

5. Writes the results of the experiment

B. The resource part of the program which:

1. Performs a sequence of user specified system calls

2. Distributes the activities "evenly" across resources

3. Allocates a specific amount of memory

4. Does specified quantities of I/O to a number of files

5. Consumes CPU time by executing a compute loop a specified number of times

Since this is the part of interest to the experimenter, it will be described in detail.

The analyst has at his disposal the following major variables:

A. A menu of system calls that can be done

B. The number of iterations desired through the compute loop

C. The amount of memory desired

D. The type and amount of I/O to be done on each file

E. Piping or transferring of messages between processes

See Appendix 1 for specific inputs and output format.

System Calls

The following system calls are supported:

1. Fork/Exec
   A Fork and Exec are done to create and execute another copy of the synthetic process. The child process created has its own resource consumption to perform. It may in turn Fork and Exec to yet a third process.

2. Kill
   A process can kill another process.

3. Nice
   A process can lower its priority by the use of this command.

4. Sleep
   A process sleeps for a specified number of seconds.

5. Wait
   A process will wait for one of its children to exit. It will continue after the first one returns.

Compute Kernel

The compute algorithm is the same as that used by Buchholz (Ref. 3), namely the summing of the cubes of integers.

$$\sum_{k=1}^{N} k^3 = \left| \frac{N(N+1)}{2} \right|^2$$

In SP, N = 10.

Input/Output

Two types of I/O are presently supported:

1. The system calls READ and WRITE

2. The character I/O subroutines, GETC, PUTC, GETW and PUTW

For each type of I/O, the user specifies:

1. The total number of bytes to be transmitted

2. The number of bytes to be transmitted in one I/O call. For the second type of I/O, that can only be one (for GETC or PUTC) or two (for GETW or PUTW).

3. An indicator that the user wants to do "reads" only, "writes" only, or 50% "writes" followed by 50% "reads". To read, the file must have been prewritten, either by an earlier experiment or some other means.

4. For READ or WRITE, the user can also do SEEK of a specified number of bytes between consecutive accesses.

All of the above I/O will be done to a user-specified file (which will be created, if need be) or to a program-generated file which will be deleted at the end of the process. The user can specify any device including the terminal.

Pipe and Messages

For both pipes and messages, the user specifies:

1. Total number of bytes

2. The number of bytes to be transmitted in one pipe call or in one send/receive

3. An indicator that the user wants to do "reads" or "writes" on a pipe or "send" or "receive" a message.

4. The process to (from) which it is piping or sending (receiving) the message.

Memory

The user can specify the amount of memory that is to be used. Any amount above what is used by the program will be allocated and then freed at the end of the experiments. The program itself is about 28000 bytes.

Prodding

One process can prod another into doing work by sending it a message. The sending process, say process 1, just sends a regular message. Each time, the receiving process, say process 2, receives the message, it goes through its compute and I/O resource loops. Process 2 then waits for the next message. To terminate the "prodded" process 2, process 1 just kill it. A child as well as a parent can both prod or be prodded.

Allowable Network Configurations

To implement SP in a reasonable manner, only certain network configurations of communicating processes are supported.

1. Only one communication path between processes is acceptable.

2. If a process is being prodded by another (by receiving its message), it cannot communicate with that process in any other way.

3. The allowable configurations are the following:

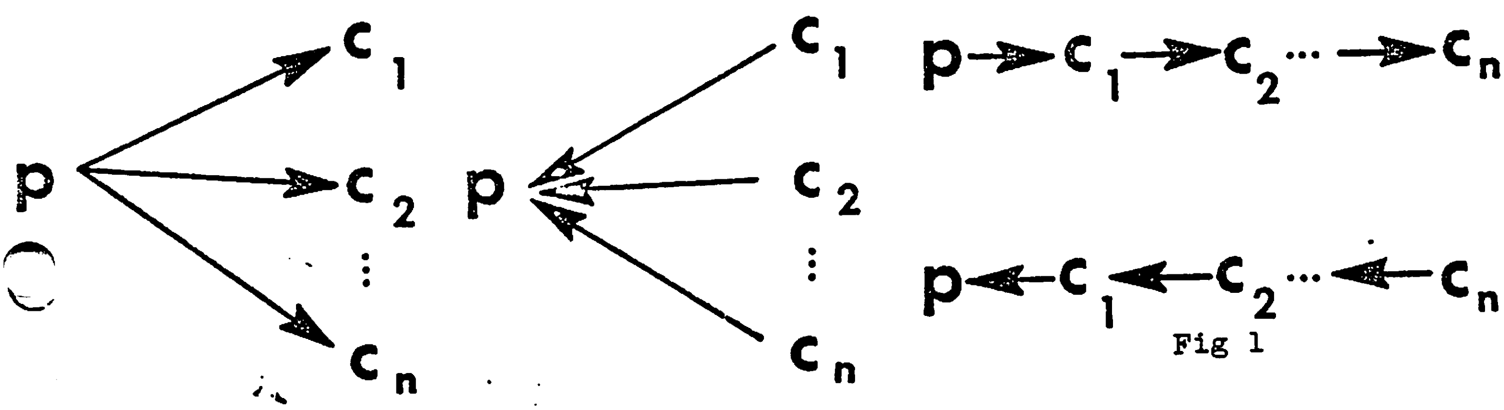

Fig 1

The above restrictions enable the synthetic process to be implemented without imposing any priority scheme. With these configurations, there should be no deadlock. An extension of SP would be to allow two-way communication by adding a priority scheme. The restrictions hold only for processes that are communicating (via pipes or messages). If the processes are all consuming resources independently (i.e. doing I/O to a file or computing), any configuration is acceptable.

Calibration

In order to use the SP in modeling, the program must be calibrated. That is, the CPU time of a compute kernel iteration and a transmission of a specific number of characters must be known for some base hardware or software configuration.* Therefore, certain calibration experiments were done. (see Appendix 2).

The timing was done using TIME and TIMES system calls. The elapsed time is measured in seconds. The process user and system times are measured in 1/60 of a second. The results are given in milliseconds.

To time using the system calls TIME and TIMES, a minimum amount of resources must be consumed. If too small a level is specified, the noise will overpower the timing measurements.

Acknowledgement

The author thanks J.E. Ritacco for his valuable assistance.

D. R. Bernstein

Att;
Ref 1to4
App 1
App 2
Table 1

* To consume a certain number of CPU seconds, say x sec, with additional I/O operations in the same job, one cannot specify i iterations through the kernel, each iteration taking k seconds and i = x/k. Rather, i = (x-n*c)/k where n is the number of characters transmitted and c is the cpu time per character.

References

1. J.F. Maranzano, A Proposal For Using A Synthetic Profile To Represent a Real Jobstream, MM-72-8234-2, March 6, 1972.

2. R.L. Klein and J.E. Ritacco, Synthetic Jobstream Model Of A Batch Workload, TM-76-8234-13, May 15, 1976.

3. W. Buchholz, A Synthetic Job For Measuring Performance, IBM Systems Journal, Volume 8, 1969.

4. R.B. Brandt, Proposal For UNIX Interprocess Communication, TM-76-8234-4, March 17, 1976.

APPENDIX 1

User Documentation For A UNIX Synthetic Process

The following is a description of the user inputs and the outputs of the program, SP.

SP can be used to model applications at a gross level of detail using the UNIX or MERT operating system where multiple processes and intercommunication between these are involved. Systems calls such as FORK's, EXEC, piping and message handling can be modeled. Only a restricted subset of these is included. With this tool, the experimenter will be able to compare:

1. UNIX pipes
2. Messages under UNIX as implemented by R.B. Brandt (Ref. 4)
3. Messages under MERT/UNIX

<u>User INPUT</u>

The inputs consist of:

1. A command with arguments that starts the execution of SP.
2. A set of parameters for each synthetic process in the network.

<u>COMMAND and ARGUMENTS</u>

%sp -t <I >O

The argument is optional

t - the user does not want any timing information. This might be invoked when other measurement tools are being used in conjunction with SP.
The standard input will contain the input for the parent process (which might be the only process). The standard output will contain the output for the parent process.

<u>INPUT DATA PARAMETERS</u>

The input consists of:

A. Optional header
B. Optional System Calls
C. Global arguments
D. 1 to 8 optional data transmission arguments
E. A delimiter

These are described in detail below.

A. Header

This is an optional free form character string which must start with a "-h" and be less than 100 characters. This can be used to identify an experiment.

B. System calls

The general format for the system call is

-s syscall number name
separated by one or more blanks.

The following system calls are supported:

1. fork file/processname

   SP will FORK and EXEC to another copy of SP. The filename and process name for each child are identical. The filename names the file where the input for the child process resides. It is also used as the name of the child process (needed if the user wants to read/write a pipe from/to it or send or receive a message).

2. kill processname

   The named process is killed. The process goes to a routine that immediately does the timing, performs needed clean-up and EXITs.

3. nice number

   SP will issue a NICE within a process.

4. prod processname

   Prod is not a literal system call. Rather it is an indication to the process that the message it is receiving should be a ´prod´ for it to consume resources. Therefore when the process receives the message, it will go through the resource usage specified on the ´g´ and ´f´ arguments (described below). All messages used for prodding will be received as the maximium message size (212).

5. sleep number

   The process will SLEEP for the specified number of seconds.

6. wait

The parent process will WAIT for any child to terminate.

The system calls can be grouped in any order that makes sense to the experiment designer. There is limited sanity checking in order to maximize flexibility.

C. Global arguments

The global resource consumption arguments will look like:

    -g NPASS NCOMP NMEM

separated by one or more blanks.

NCOMP : number of iterations through compute loop
The compute loop code generates 137 assembly instructions.

NPASS : number of passes through totality of resource consumption defined

NMEM : total memory requirements in bytes (optional)

If the user requests a memory allocation less than the size of the program (about 28000 bytes), the NMEM parameter will be ignored and no additional memory will be allocated. There can only be one set of global arguments per experiment.

D. Data transmission arguments

This will specify the I/O, piping and message sending/receiving that will be done. For each instance of data transmission specified, the user will supply a set of arguments. There is a maximum of 8 per process. The file arguments will be in the following order:

    -f TYPE NBYTE LBYTE IOIND SBYTE DATATARGET

TYPE : Type of data transmission to be done
Values can be 1 through 4.
1. Standard UNIX. (GETC, PUTC, GETW and PUTW)
2. Low level I/O. (READ and WRITE)
3. Pipe (Read or Write on a pipe)
4. Message (Send or Receive a message)

NBYTE : Total number of bytes transmitted in conjunction with this target (file,pipe or message).

LBYTE : Number of bytes to read or write per system call
For IOTYPE 1, LBYTE = 1,2

For IOTYPE 2, LBYTE > 0
for IOTYPE 3, LBYTE <= 4096
for IOTYPE 4, LBYTE <= 212

IOIND : Read, write or both indicator. Values can be 0,1,2.
0 = read only (receive if IOTYPE = 4)
1 = write only (send if IOTYPE = 4)
2 = write for half the NBYTEs then read for the other half. This is not supported for pipes or messages.

SBYTE : This parameter is valid only if the user is doing READS and WRITES (IOTYPE = 2). This will indicate that a SEEK should be done in addition to reading and/or writing the file and by how many bytes on the average. This is an <u>optional</u> parameter. If no number is specified, no SEEK will be done.

DATATARGET : For <u>I/O</u>, a file is named by the user. SP uses that file to do the specified I/O . This is an <u>optional</u> parameter. A scratch file will be provided if none is specified. If the user wants to do I/O directly onto the terminal, "term" must be specified as the file name.
For TYPE 3 or 4, a process name (rather than a file name) must be specified. The process name for a child is the same as the name of its input file. The parent, whose input comes from the Standard Input, is called "parent".

If a process is "prodded", it must not have an -f argument for receiving the message.

E. End indicator
-e
-e will indicate the end of input for data transmission arguments. This can be followed by more sequences of -h,-s,-g,-f, and -e.

<u>Definition of Experiment</u>

A set of input (between end indicators) will be timed and will constitute an experiment. However, all system calls will be done strictly in the order that they appear and no effort will be made to clean up after them. For example, if there is a Fork to process A by the parent, process A will be alive until it either finishes and EXITS, or is killed. If the parent forks to process A again while the original A is still alive, unpredictable results will occur.

OUTPUT

The user output is found in the Standard Output file for the parent process (which might be the only process). For the children processes, the file, spout||input filename (the literals spout concatenated with the input file name for the child) is created by the program. The total file name cannot be bigger than 14 characters. It contains the output for the child.

For each process, the output will consist of:

1. All input including any default parameters per experiment
2. Elapsed time, user CPU and system CPU time for each experiment. This is obtained by having the program call TIME, and TIMES (Unix system calls) at the beginning and at the end of all passes through the resource usage.
3. Total elapsed time, user CPU and system CPU time. This is the sum of all times in (2) above.

Example 1 Input

```
/*  pipe example */
%sp

-h parent process                       /* parent process input */
-s fork c1a                             /* fork and exec to process
                                        name c1a */
                                        /* input for process name c1a
                                        is in file c1a */
-s nice 4                               /* parent process
                                        lowers its priority */
-g  1 1000                              /*  global card */
-f  3 1000 100 1 c1a                    /*  write on a pipe to c1a */
-e

/*This is the child (c1a) input contained in file c1a  */

-h  child will read pipe
-g  1 100
-f  3 1000  100 0 parent                /*  reads the pipe */
-e
```

Example 1 Output

```
 parent process

system calls
1 fork  cla 0
2 nice   4
npass = 1 ncomp = 1000.000000 nmem = 0.000000
file    iotype      nbyte   lbyte     ioind     sbyte     file/process

 1         3          1000    100        1          0      cla
time taken * in milliseconds*
real time = 1000.000 usertime = 368.000 system time = 32.000
** total ** time taken in milliseconds
real time = 1000.000 usertime = 368.000 system time = 32.000


 child will read pipe

npass = 1 ncomp = 100.000000 nmem = 0.000000
file    iotype      nbyte   lbyte     ioind     sbyte     file/process

 1         3          1000    100        0          0      parent
time taken * in milliseconds*
real time = 0.000 usertime = 32.000 system time = 16.000
** total ** time taken in milliseconds
real time = 0.000 usertime = 32.000 system time = 16.000
%
```

Example 2 Input

```
/*  message example */
$sp

-h  parent will prod child
-s  fork  c2a                          /*  fork and exec to process c2a */
-s  fork c2b                           /*  fork and exec to process c2b */
-s  sleep 5
-g  1 1000
-f 4 424 212 1 c2a                     /* send two messages to c2a */
-f 4 848 212 1 c2b                     /* send 4 messages to c2b */
-e
-s  kill c2a
-s  kill  c2b
-e
/* input for c2a contained in file c2a */
-h  c2a is being prodded
-s prod parent                         /* each message received from parent wil
                                       the process write 51200 bytes */
-g  1 0
-f 2 51200 512 1 c2afile               /* write 51200 bytes */
-e

/*  input for c2b contained in file c2b */

-h  c2b is being prodded
-s prod parent                         /* parent is prodding c2b */
                                       /*each message received by c2b will make
                                       it go through the compute kernel
                                       1000 times */
```

-ρ 1 1000

-ε

Example 2 Output

```
PARENT WILL PROD CHILD

SYSTEM CALLS
1 FORK  C2A 0
2 FORK  C2B 0
3 SLEEP  5
NPASS = 1 NCOMP = 1000.000000 NMEM = 0.000000
FILE    IOTYPE      NBYTE   LBYTE    IOIND     SBYTE    FILE/PROCESS
                                                        C2A
 1        4          424     212       1         0      C2B
 2        4          848     212       1         0
TIME TAKEN * IN MILLISECONDS*
REAL TIME = 9000.000 USERTIME = 540.000 SYSTEM TIME = 48.000
SYSTEM CALLS
1 KILL  C2A 0
2 KILL  C2B 0
NPASS = 0 NCOMP = 0.000000 NMEM = 0.000000
 * C2B-
 - C2A-
.PROCESS CAUGHT HANG UP
 PROCESS CAUGHT HANG UP
IME TAKEN * IN MILLISECONDS*
REAL TIME = 0.000 USERTIME = 16.000 SYSTEM TIME = 0.000
** TOTAL ** TIME TAKEN IN MILLISECONDS
REAL TIME = 9000.000 USERTIME = 556.000 SYSTEM TIME = 48.000
% CAT SPOUTC2A SPOUTC2B


 C2A IS BEING PRODDED

 SYSTEM CALLS
 1 PROD  PARENT 0
 NPASS = 1 NCOMP = 0.000000 NMEM = 0.000000
 FILE    IOTYPE      NBYTE   LBYTE    IOIND     SBYTE    FILE/PROCESS
                                                  0      C2AFILE
  1        2         51200    512       1
 TIME TAKEN * IN MILLISECONDS*
 REAL TIME = 10000.000 USERTIME = 32.000 SYSTEM TIME = 992.000
 ** TOTAL ** TIME TAKEN IN MILLISECONDS
 REAL TIME = 10000.000 USERTIME = 32.000 SYSTEM TIME = 992.000


  C2B IS BEING PRODDED

  SYSTEM CALLS
  1 PROD  PARENT 0
  NPASS = 1 NCOMP = 1000.000000 NMEM = 0.000000
  TIME TAKEN * IN MILLISECONDS*
  REAL TIME = 9000.000 USERTIME = 1216.000 SYSTEM TIME = 48.000
  ** TOTAL ** TIME TAKEN IN MILLISECONDS
  REAL TIME = 9000.000 USERTIME = 1216.000 SYSTEM TIME = 48.000
  %
```

Shell Procedure

To combine all the output from a network of processes in a particular file and delete the individual output files, the following shell procedure can be used.

```
sp <$1 >spout$1
cat spout* >$2
rm spout*
```

```
sh spcat a b
```

is used to execute the above shell
where a is the input file for the parent process
and b will be the output file for all the processes.

Appendix 2

The experiments performed to calibrate the SP program are the following:

1. Exercise the compute kernel (with no I/O)
2. PUTC, GETC I/O TYPE =1 with LBYTE =1
3. READ/WRITE I/O TYPE = 2 with LBYTE = 512
4. READ/WRITE to a PIPE I/O TYPE = 3 with LBYTE = 4096
5. RECEIVE/SEND MESSAGE I/O TYPE = 4 with LBYTE = 212

Each set of experiments was performed at least 3 times and the average is given.

Calibration Experiments [in msec]						
CALL	UNIX 11/70		UNIX 11/45		MERT 11/45	
	user t.	sys.t.	user t.	sys. t.	user t.	sys. t.
comp. kernel	.275	~0	.673	~0	.637	~0
PUTC/byte	.0741	.0066	.147	.0162	.134	.0170
GETC/byte	.0655	.0092	.138	.0109	.121	.0258
WRITE 512/call	~0	3.467	~0	6.080	~0	12.757
READ 512/call	~0	3.435	~0	6.485	~0	12.181
PIPE WRITE 4096	~0	17.665	~0	32.97	~0	103.83
PIPE READ 4096	~0	18.456	~0	37.243	~0	95.05
MSG SEND 212	-	-	~0	2.16	-	-
MSG RCV 212	-	-	~0	2.56	-	-

Table 1

This table should be taken as an illustration of the use of the tool, SP. The relative values of the measurements should be used only as a starting point into an investigation of the different systems.