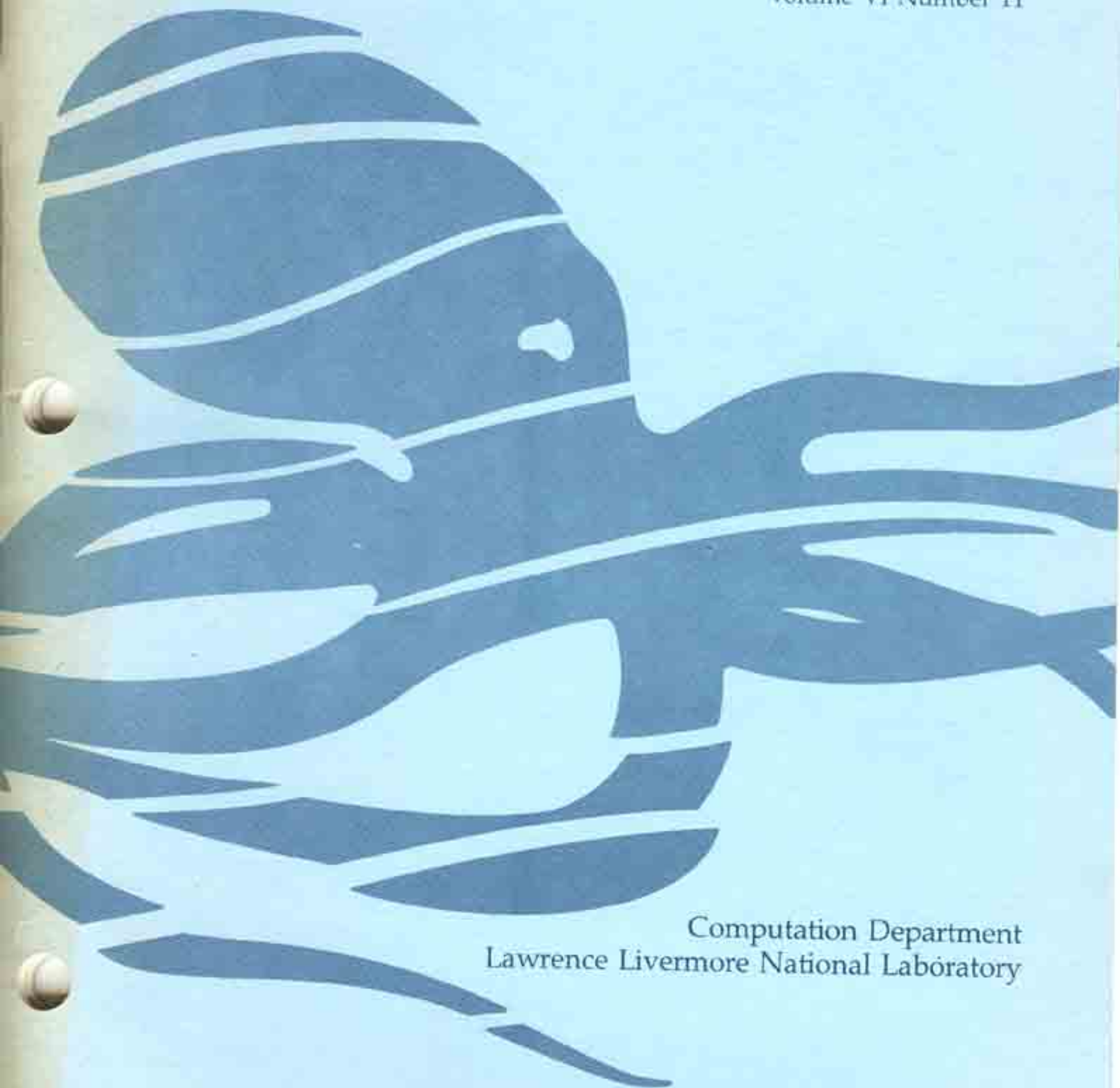


TENTACLE

December 1986
Volume VI Number 11



Computation Department
Lawrence Livermore National Laboratory

LOCATOR

<u>NAME</u>	<u>EXT.</u>
CMRD Consulting Office (Hours: 09:00-12:00, 13:00-16:00)	3-2976
Computer Documentation Library	2-0592
Computer Education and Training	2-4257
Coordination Center	2-4531
LANL Software Consultants	114*1*667*5745
LCC Document Coordinator	2-4311
LCC Software Consultants (Hours: 09:00-12:00, 13:30-16:30 Closed Tuesdays)	2-3724
Network Operator (RJETS)	2-3732
NMFEC Software Consultants	2-1544
Tape Librarian	2-3734
<i>Tentacle</i> Distribution	2-4257
Terminal Repair	2-3718
TID Library	2-5277
TMDS Service	2-8882
User Services	2-3705

Tentacle Schedule

	<u>Issue</u>		
	<u>January 1987</u>	<u>February 1987</u>	<u>March 1987</u>
Authors submit articles*	December 1	January 5	February 2
Editing/author review	December 2-8	January 6-13	February 3-10
Production, Layout	December 9-17	January 14-19	February 11-18
Div., Dept., D.O. reviews	December 18-19	January 20-21	February 19-20
Last chance to make time-critical or technical changes-	December 19, noon	January 21, noon	February 20, noon
To printing	December 22	January 22	February 22
Distribution	January 5	February 5	March 5

*Long articles that are not time-critical may be held until the following month to ensure adequate time for editing. Therefore we recommend that articles longer than 4 pages (200 lines) be submitted before this date if possible.

TENTACLE

December 1986
Volume VI Number 11



Tentacle is published monthly by the User Systems Division, Computation Department, at Lawrence Livermore National Laboratory, Livermore, California.

To submit an article for publication, contact the Scientific Editor (Ext. 2-4311), Octopus address: MAIL: *karl*, network address: *karl@lll-erg.ARPA*. Articles may be submitted on a Macintosh disk, IBM disk (ASCII file), or you may write an online TRIX RED file *filename* to the *Tentacle* directory:

```
XPORT
WR .903904:TENTACLE:filename
END
```

Scientific Editor
Karl Dusenbury

General Editor
Elsa Pressentin

Editorial Assistant
Jane Winter

Editorial Staff
Lila Abrahamson
Harriet Kroopnick
Gary Shaw
Terry Williams

Distribution
Christa Sobczak
Ext. 2-4257

Contents

SYSTEM/SOFTWARE NEWS

Multiprocessing—Timing Methods (<i>Gary J. Blair</i>)	1
NLTSS—NLTSS Memory Management (<i>Robert E. Strout II</i>)	11
CMRD Reports—MATHLIB vs NMATHLIB (<i>Tokihiko Suyehiro</i>)	16
Latest from LabNet (<i>George Pavel</i>)	17

WORKSTATIONS

PC Corner (<i>Rich Serbin</i>)	
Is Your Macintosh or IBM PC a Security Violation?	20
Storing Classified Data on Disk	20
A Warning about Univation's Slimline Removable Disk Unit	21

ISSUES AND ANSWERS

Consulting Office Commentary—Vulnerability of LTSS	
SMM Control Words (<i>Maurie Louis</i>)	22
Selected Octograms	30

ELECTRONIC PUBLISHING

Producing the <i>Tentacle</i> (<i>Elsa Pressentin</i>)	34
MacΣqn for Setting Equations (<i>Jane Winter</i>)	37
Getting Files from the PC to the Mac (<i>Jane Winter</i>)	38

DIVERSIONS

Puzzles (<i>Henry Larson</i>)	39
Computist's Corner—Geometrical Atoms (<i>R. K. Cralle</i>)	41
The Octopus's Garden (<i>Joanne Perra and Neale Smith</i>)	43

From the Editor's Desk

We have a full issue this month. NLTSS author Rob Strout indicates that the proposal outlined in his article affects all user-supplied memory management under NLTSS. Another important article is *PC Corner*; Rich Serbin discusses computer security as it relates to file manipulation between mainframes and personal computers.

Hurrah! *The Octopus's Garden* is back this month; see page 43. Also in the Diversions section: solutions to both the October and November *Puzzles*.

We hope you enjoy this issue. And until next year, Happy Holidays!

Multiprocessing

TIMING METHODS

Gary J. Blair

In last month's article,¹ we discussed two ways to partition our code and learned about several mechanisms for synchronizing tasks. Now is a good time to step back and ask: Does all this stuff buy us anything—or is it just an exercise in futility?

Though we fervently hope that we are indeed gaining something, this question must be answered on a program-by-program basis. To answer the question, we need performance measurements.

In this article we discuss several library routines that enable us to measure the performance of our codes. We first develop a computational model that aids in understanding the timing primitives. We then explain the timing routines and present short examples illustrating their usage. We end by presenting two routines that allow tasks and processes to obtain their own identities.

A computational model

The following model is useful for distinguishing between task, process, and processor and will aid in understanding the available timing routines. Other useful computational models may be found in Refs. 2 and 3.

Suppose that we wish to perform all of Mozart's operas (I believe there are twelve) in their entirety, as quickly as possible. This corresponds to a job, or the single execution of a program. Because each opera is an independent entity, it is possible that some or all of the operas may be performed in parallel. Thus, the individual operas are analogous to tasks. Suppose we have a second job, which is to perform all of Beethoven's nine symphonies, again as quickly as possible. Here, the individual symphonies correspond to tasks, and two or more of them may be performed simultaneously. There are a number of orchestras available to perform these pieces, and a number of music halls in which the orchestras play. The music halls represent processes, and the orchestras are the analogs of processors. These associations reflect the fact that a job is a collection of tasks, a task is work to do, a process is a work environment, and a processor actually performs the work. To maintain the analogy, we need the following rules.

For a job to be executed, it is assigned a set of music halls in which its various musical pieces will be performed. However, once a music hall has been used to perform a Beethoven symphony, it may not be used to play any of Mozart's operas, and vice versa. This reflects the fact that processes are associated with particular jobs. Remember: performing the Mozart operas and the Beethoven symphonies are two distinct jobs.

In order to be performed, a piece of music (task) must be assigned to a music hall (process). Once this is done, we must wait for an available orchestra (processor). When both these conditions are met, the orchestra moves into the music hall and begins performing the piece. Beethoven symphonies may be played simultaneously in two or

more music halls, but they must be different symphonies; it is illegal for the same symphony to be played in different music halls at the same time. This also applies to Mozart's operas.

Now, it may be necessary to schedule an intermission during a performance for the comfort of the audience. At the intermission, the orchestra may move into another music hall and begin performing the piece assigned to that other hall. Eventually the performance in the music hall where the intermission occurred will resume. The orchestra may or may not be the same one that was there before, but it will continue the same piece, at the point where it was suspended for the intermission. This corresponds to a process being preempted by the operating system.

Finally, the audience in a particular music hall may become bored with the selection being performed there. In this case, a different musical piece, still part of the same job, is assigned to the music hall. The piece that had become boring may be reassigned to another hall associated with the same job, in which case it will be performed starting at the point where the previous audience became bored, not from the very beginning. This is analogous to what happens when a task executes a synchronization operation and is left in a suspended state.

Timing routines

A number of routines are available that enable users to measure the performance of their codes. We discuss them here and present an example of the usage of each.

IZM00

IZM00 is a BASELIB and NBASELIB routine that returns a variety of timing information.⁴ IZM00 is invoked as a function in the following manner:

```
integer      timeInfo(5)
integer      error
...
error = izm00 (timeInfo)
```

If the call returns normally, `error` will have a zero value, and `timeInfo` will have the following values:

<code>timeInfo(1)</code>	Total CPU time used, in microseconds
<code>timeInfo(2)</code>	Total I/O time used, in microseconds (LTSS only)*
<code>timeInfo(3)</code>	Total system time used, in microseconds (LTSS only)*
<code>timeInfo(4)</code>	0
<code>timeInfo(5)</code>	0

In terms of our model, `timeInfo(1)` is the sum of the times that all orchestras have spent performing pieces from a particular job. For example, assume two orchestras began playing Beethoven symphonies at 1:00 p.m. The first orchestra, playing the Fifth Symphony, stops at 2:00 p.m., but the second orchestra, playing the Ninth Symphony,

*Currently, the NBASELIB version returns zero for `timeInfo(2)` and `timeInfo(3)`.

continues until 3:00 p.m. At this time, a call to IZM00 will return a `timeInfo(1)` value of 3 hours (represented in microseconds), the sum of the times the two orchestras have played.

We omit an example of the usage of IZM00, as its usage is quite similar to the usage of SECOND, which we discuss next.

SECOND

SECOND is a FORTLIB routine that returns the cumulative CPU time used by the job, in seconds instead of microseconds.⁵ SECOND calls the BASELIB routine IZM00 discussed above, and in effect it returns the value `timeInfo(1) × 106`. However, SECOND is more convenient to use, unless you need to obtain I/O and system charges or are simply trying to avoid FORTLIB. A typical usage of SECOND is shown below.

```
timeBefore = second (timeBefore)      $$$ Measure overhead.
timeAfter  = second (timeAfter )
overhead   = timeAfter - timeBefore

timeBefore = second (timeBefore)      $$$ Now measure work.
...                                              $$$ Perform work here.
timeAfter  = second (timeAfter)
timeUsed   = timeAfter - timeBefore - overhead
```

We make the overhead calculation in order to determine how much the call to SECOND itself costs. Once we know this, we can accurately determine the cost of the actual computation we wish to perform. The computation may be a loop, a subroutine, and so on.

SECOND works well in a monoprocessing code, i.e., where we have only one orchestra playing at any given time. However, it may yield false results in the multiprocessing world. Consider the example we discussed above, where two orchestras both begin playing Beethoven symphonies at 1:00. Assume we are trying to determine how long it takes to play the Ninth Symphony. We attempt to measure by making the call:

```
timeBefore = second (timeBefore)
```

at 1:00, when the two orchestras begin playing, and the call

```
timeAfter  = second (timeAfter)
```

at 3:00, when the second orchestra stops playing the Ninth. We then subtract these two quantities, and subtract the overhead. Unfortunately, the value thus produced (3 hours) is not an accurate measurement, because it includes the hour that the first orchestra was playing the Fifth Symphony. To obtain an accurate result using SECOND, users must ensure that no other processors are working on the job while the measurement is being taken, including during the overhead measurement.

SECOND may also be called as a subroutine. The following calls are equivalent.

```
timeBefore = second (timeBefore)
call second (timeBefore)
```

TQTSKTIM

As we have just seen, `SECOND` is useful for timing multiprocessing codes, but may produce erroneous results when multiprocessing. The `NSYSLIB` routine `TQTSKTIM` allows us to overcome this problem and obtain accurate timing information while multiprocessing. `TQTSKTIM` returns the amount of time used by the calling task; no time for any other task is measured by the call. `TQTSKTIM` is invoked as an integer function and returns the number of CPU cycles used by the task. The following code skeleton illustrates how to measure the time a task spends performing a computation.

```

Integer    cyclesBefore, cyclesAfter, cyclesUsed, overhead
Integer    tqtsktim
Integer    dummyArg                $$$ For call to TQTSKTIM
                                      $$$ The argument may be omitted
                                      $$$ with no effect on the result.

cyclesBefore = tqtsktim (dummyArg)    $$$ Measure overhead.
cyclesAfter  = tqtsktim ()            $$$ An alternate form.
overhead     = cyclesAfter - cyclesBefore

cyclesBefore = tqtsktim (dummyArg)    $$$ Now measure work.
...                                                $$$ Perform computation
cyclesAfter  = tqtsktim (dummyArg)
cyclesUsed   = cyclesAfter - cyclesBefore - overhead

```

Let us once again think of our musical model. Consider Mozart's opera *The Magic Flute*. Each orchestra may have spent some of its time performing this opera, and we don't care in which music hall the performing was done. If we sum up these times for all the orchestras, we have the total time any orchestra spent performing the opera. This is the number `TQTSKTIM` returns.

The `NLTSS` version of `TQTSKTIM`, found in `NSYSLIB`, works correctly. However, the `LTSS` version in `SYSLIB` currently returns incorrect information and should not be trusted.

TQCYCLES

`TQCYCLES` determines how much time a particular process has used. `TQCYCLES` is called as an integer function and returns the time used, in cycles.

```

Integer    procNum                $$$ Process number we want information
                                      $$$ about.
Integer    retryCount             $$$ Number of tries to get consistent
                                      $$$ value.
Integer    cycles
Integer    tqcycles
...
cycles = tqcycles (procNum, retryCount)

```

`TQCYCLES` returns the number of attempts required to get consistent timing information in its second argument, `retryCount`. We are not usually interested in this value.

The first argument to TQCYCLES is `procNum`, which specifies the process we want timing information about. On the Cray X-MP/48, legal values of `procNum` are -1, 0, 1, 2, 3, and 4. If `procNum` is 0, 1, 2, 3, or 4, it specifies a particular music hall, and we are in effect asking the question "How much time was spent by all orchestras in the specified music hall?" We don't care what selections were being played, we just want the sum of the times spent by all the orchestras. TQCYCLES returns this value.

The previous paragraph suggests a limitation we have not talked about before: each job is allowed a maximum of five processes. Now, we didn't just pull the number 5 out of thin air. It is equal to the number of processors on our four-processor Cray X-MP/48, plus 1. But why? By choosing this number, we allow one working process for each processor, and one to handle console interrupts (these are not yet implemented). The process number of a monoprocessing job, or the number of the initial process in a multiprocessing job, is zero.

Let's return to our discussion of TQCYCLES. If `procNum` is -1, TQCYCLES returns the job residence time. In terms of the model, the job residence time for playing all the Beethoven symphonies is the amount of time any symphony has been performed in any music hall. If two or more pieces are being performed simultaneously during some time period, that time period is counted only once. Job residence time is more easily understood with a diagram. The timeline in Fig. 1 shows the times when processes in a fictitious job are active.

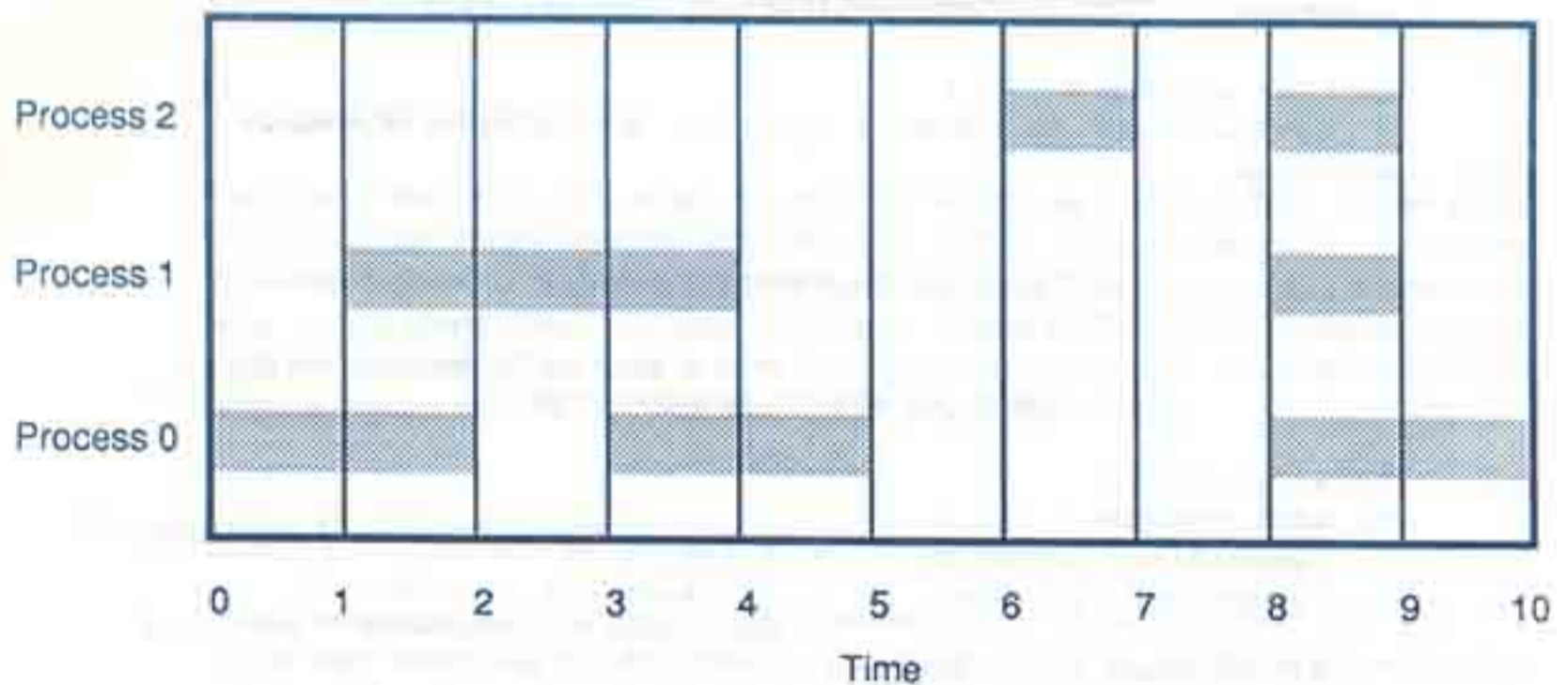


Fig. 1. Timeline showing the active periods of the three processes on a fictitious job.

Consider the timeline in terms of our model, where we are measuring the time to play Beethoven's symphonies. We can see that between time 0 and time 2 a symphony—we don't know or care which one—was being played in music hall 0. We also see that a symphony was being played in hall 0 between times 3 and 5, and between times 8 and 10. Similarly, a symphony was played in hall 1 between times 1 and 4, and again

between times 8 and 9, and in hall 2 between times 6 and 7 and times 8 and 9. The time periods from 0 to 5 contribute a 5 to the job residence time, since at least one music hall was active at any moment during this entire time. Similarly, the time period from 6 to 7 contributes a 1, and 8 to 10 contributes a 2. Therefore, the overall job residence time is $5 + 1 + 2$, which is 8. TQCYCLES will return this value when called with `procNum` set to -1. An alternative way to think about this is to consider the number of available time units (10) and subtract the number of time units during which *no* process was active (2), yielding the job residence time (8).

TQCYCLES is useful for measuring the effective overlap you obtain by multiprocessing your program. We compute the total time used by all the processes, then divide this by the job residence time. The following skeleton illustrates how to calculate your overlap.

```

integer  cyclesBefore(-1:4), cyclesAfter(-1:4),
>        cyclesUsed(-1:4)
integer  overhead, totalCycles
integer  procNum, retryCount
integer  tqcycles

        $$$ We assume multiprocessing at this point
        $$$ to enable accurate measurements.

cyclesBefore(-1) = tqcycles (-1, retryCount)
cyclesAfter(-1)  = tqcycles (-1, retryCount)
overhead         = cyclesAfter(-1) - cyclesBefore(-1)

do bfor procNum = -1, 4
    cyclesBefore (procNum) = tqcycles (procNum, retryCount)
bfor  continue

...
...        $$$ Perform the multiprocessing computation.
...

        $$$ Again we assume multiprocessing

totalCycles = 0
do aftr procNum = -1, 4
    cyclesAfter (procNum) = tqcycles (procNum, retryCount)
    cyclesUsed  (procNum)  = cyclesBefore (procNum) -
>        cyclesAfter (procNum) - overhead
    if (procNum .ne. -1) then
        totalCycles = totalCycles + cyclesUsed (procNum)
    endif
aftr  continue

        $$$ Calculate the effective overlap.

overlap = float (totalCycles) / float (cyclesUsed(-1))

```

If we multiprocessing during the timing loops, we may obtain incorrect results. Other processors may be working on the job while the measurements are being taken, and the cycles they contribute may not be counted. Consequently, to obtain accurate measurements we must monoprocess during these loops. The timing loops themselves require a few cycles to execute, and not all the cycles are counted. Therefore, the overlap

that we compute is not entirely accurate. However, this is a problem only if we are measuring extremely small amounts of work. If the amount of work we are measuring is large compared to these few uncounted cycles, the computed overlap will be sufficiently accurate for most purposes.

On the four-processor Cray X-MP/48, the maximum possible overlap is 4. Now, for multiprocessing to be worthwhile, we want to see an overlap as close to 4 as possible. Deciding whether a given overlap is acceptable, however, is very subjective. One user may be happy with an overlap of only 1.5, whereas another may be unwilling to settle for an overlap less than 3.8. Some codes may be inherently highly sequential. Their running times may not improve much, even though programmers make great efforts to parallelize them. Other codes may be quite amenable to multiprocessing and may yield large effective overlaps in response to a very small programmer effort. Once again, the final decision as to what is a worthwhile overlap is up to you, the programmer.

Note that you can compute your code's maximum (ideal) overlap on the Cray X-MP/48 only if you have all four processors dedicated to your job. Ideal overlap cannot be accurately measured during timesharing because other jobs are contending for the processors. Thus, one of your job's processes may have to wait for a processor that is being used by another job. This may result in an inaccurate measurement of maximum overlap.

TQCYCLES, like TQTKSTIM, works correctly only under NLTSS (NSYSLIB). The LTSS version in SYSLIB currently returns incorrect information.

The cost of timing calls

The cost of the timing calls is not the same for NLTSS as it is for LTSS. Under LTSS (SYSLIB), the library routines SECOND and IZM00 make system calls. These are relatively expensive. However, the time spent executing these system calls shows up in your system charge time, not your CPU time. Under NLTSS (NSYSLIB), no system calls are involved; all the work is done directly by library routines. Therefore, your CPU usage is greater, even though your total charge time will be smaller.

Overcoming the identity crisis

Like a person, a task or a process may sometimes have an identity crisis; that is, it may not be sure which task or which process it really is. However, unlike people, who often stew about this problem for months or years, tasks and processes can quickly ascertain their identities by using the NSYSLIB routines TQTSKNUM and TQPRCNUM.

TQTSKNUM

TQTSKNUM is an integer function that returns the task number of the calling task. The routine is invoked in the following manner:

```

integer  tqtsknum          $$$ The library routine.
integer  dummyArg         $$$ Dummy argument to thefunction.
integer  myTaskNum        $$$ My task number.
...
myTaskNum = tqtsknum (dummyArg)
...

```

The task number is assigned when the task is started via the TSKSTART routine, and never changes. The task number of the original task in a job is 1. Other task numbers are the consecutive positive integers beginning with 2. These are assigned to tasks in the order in which the tasks are created. In terms of our musical model, this function invocation is equivalent to one of the musical selections asking the question, "What piece am I?" and receiving the answer, "You are Mozart's opera *Don Giovanni*."

This routine is handy for summarizing the amount of time each task requires. Such a summary is useful in determining if we have evenly divided the work among the tasks. For instance, assume we have a number of identical tasks. If all of them require about the same amount of time to complete their work, we have a good load balance. However, if some tasks take significantly longer than the others, we have an imbalance that may result in poor performance. Following is the skeleton of a task that reports its time usage.

```

subroutine  SomeSub
integer    timeBefore, timeAfter, timeUsed, overhead
integer    myTaskNum, dummyArg
integer    tqtsknum, tqtsktim
...
myTaskNum = tqtsknum (dummyArg)          $$$ Get my task number.
timeBefore = tqtsktim (dummyArg)        $$$ Compute overhead.
timeAfter  = tqtsktim (dummyArg)
overhead   = timeAfter - timeBefore

timeBefore = tqtsktim (dummyArg)        $$$ Time the computation.
...                                           $$$ Perform computation
...                                           $$$ here.
timeAfter  = tqtsktim (dummyArg)

timeUsed   = timeAfter - timeBefore - overhead

write (unitNum,fmt1) myTaskNum, timeUsed
fmt1  format ('Task ',i20,' used ',i20,' cycles.')
return
end

```

TQPRCNUM

TQPRCNUM is an integer function that returns the identity of the currently running process. We invoke the routine as follows:

```

integer    tqprcnum                $$$ The library routine.
integer    dummyArg                $$$ Dummy argument to the function.
integer    myProcNum               $$$ My process number
...
myProcNum = tqprcnum (dummyArg)
...

```

In terms of our model, calling TQPRCNUM is equivalent to one of our musical selections asking the question, "In which music hall am I now being played?" and receiving the answer, "You are now being played in music hall 3." The value returned by TQPRCNUM is between 0 and 4 inclusive, since those are the only legal process numbers. TQPRCNUM allows a task to find out how much time the process it is running in has used. We illustrate this below.

```

integer    tqprcnum
integer    tqcycles
integer    dummyArg
integer    myProcNum
integer    retryCount
integer    cyclesUsed
...
myProcNum = tqprcnum (dummyArg)
...
cyclesUsed = tqcycles (myProcNum, retryCount)
write (unitNum,fmt1) myProcNum, cyclesUsed
fmt1      format ('Process ',i3,' has used ',i20,' cycles.')
```

Now, recall that the audience in a particular music hall may become bored with the piece being played there, in which case the piece may be assigned to another music hall. This corresponds to a task executing a synchronization call and being placed in the suspended state. If this happens, when the task resumes execution it may be assigned to a different process. Thus, if a task executes synchronization instructions between any two calls to TQPRCNUM, the two calls may yield different results. The importance of this is that the above code fragment may not perform as we desire.

What we want the code fragment to do is to determine the amount of time used by the currently executing process. However, if any synchronization instructions are executed between the calls to TQPRCNUM and TQCYCLES, the task may suspend and may be later awakened and assigned to a *different* process. In this case, myProcNum no longer contains the number of the currently executing process. Therefore, the TQCYCLES call yields timing information for a process that we are not interested in. Note that, in addition to synchronization instructions coded directly by the user, synchronization instructions may be executed during various library calls—for instance, during formatted I/O instructions.

Summary

We have provided a simple model of multiprocessing that is useful in understanding timing routines. We have explained the functionality of the available timing routines and illustrated the use of each. We hope users find this information useful in measuring the performance of their codes. As always, feel free to call if you would like additional information.

References

1. Gary J. Blair, "Synchronization Mechanisms and Partitioning Programs," *Tentacle 6* (10), 3-11 (November 1986).
2. Gary J. Blair, *Reentrancy and Multitasking on Cray Computers*, Lawrence Livermore National Laboratory, Livermore, CA, report UCID-30199 (January 9, 1985).
3. Kent Crispin, "Multitasking, Multiprocessing, and the NLTSS Multitasking Kernel," *Tentacle 6* (9), 6-16 (October 1986).
4. Arlene Getchell, *BASELIB—A Set of Basic System Functions for the CDC 7600 and Cray Computers*, Lawrence Livermore National Laboratory, Livermore, CA, report LCSD-404, Rev. 4 (August 14, 1986).
5. Barbara Atkinson, *FORTLIB—A Standard Fortran Library for the CDC 7600 and Cray Computers*, Lawrence Livermore National Laboratory, Livermore, CA, report LCSD-406 (June 14, 1982).

Gary Blair is a computer scientist in the Language Group, User Systems Division (USD).

NLTSS

NLTSS MEMORY MANAGEMENT

Robert E. Strout II

This article presents a proposal for supplying a basic memory-management scheme for use under NLTSS. Although it is centered around the current System Memory Management (SMM) routines,¹ *this proposal affects all user-supplied memory management under NLTSS.*

First, I will present a bit of history behind the current memory-management scheme supplied under LTSS and explain why that scheme is insufficient for the NLTSS environment. I will then provide a high-level description of the memory-management proposal.

As an additional concern, I also present here a condition that can result from the presence of the heap in an application code loaded for the Cray X-MP/48.

Proposal for NLTSS memory management

A bit of history

When the System Memory Management (SMM) routines were originally proposed in early 1983, other memory-management packages (as well as ad hoc memory management) were being used by many application codes. The package most heavily used by the application codes was and still is MMLIB.² This package made available a large amount of memory-management support features useful to the large application codes. However, the abundance of features carried with them a cost of space and time. SMM was proposed as a set of basic memory-management routines to be used by compilers, libraries, and utilities which did not need the extra functionality supplied by MMLIB. By supplying only basic routines, it was possible to implement the SMM with lower costs in space and time.³

A memory-management library is usually written with the assumption that it controls all the free memory space of the application in which it is loaded. If two (or more) memory managers are used in the same application code, they will be unaware of each other's presence, and each will assume it has control over the free memory. This can lead to the same memory space being allocated (and hence used) for different purposes. Since SMM was designed for use by the support libraries, it was likely that SMM would be present in an application that used MMLIB or another memory-management package.

The primary problem that had to be addressed was how SMM could coexist in the same application with another memory manager (e.g., MMLIB or ad hoc). At the time, it was judged very desirable to let the user's memory manager have control over free memory space. Thus SMM was implemented with the notions of *restricted* and *unrestricted* heaps.

When the application was loaded with an *unrestricted* heap, SMM assumed it had control over the free memory space of the code. Whenever an allocation request was made to SMM that could not be fulfilled from the current heap, SMM would expand the program's

field length and place the newly created memory space under its free-memory control. The allocation request could then be satisfied from the new memory space.

For an application loaded with a *restricted* heap, SMM assumed that it only had control over the heap space that it originally started with. When an allocation request failed to be fulfilled from this space, SMM would request the needed space by calling the subroutine MZGET. This routine is used to request the needed space from the user's memory manager. Generally, the user's memory-management package was responsible for providing the MZGET routine. This gave the application code the ability to either supply SMM with the needed space (which then became controlled by SMM) or let it error.

This has been a workable solution for a number of years. However, it has not been without drawbacks. In many cases usage of the SMM routines is invisible to the application program (i.e., they are called by other library routines). The application programmer, not aware of the usage, can load an application without a restricted heap. This can lead to problems with dual usage of memory that are not easy to detect.

To date, the support environment (compilers, utilities, libraries, and run-time support) have made the most use of the SMM, on both NLTSS and LTSS. Under NLTSS, almost all support libraries make use of SMM routines. Under both NLTSS and LTSS, the CIVIC compiler can now generate code to call SMM routines for the allocation of vector temporaries. The C and MODEL languages tie into the SMM routines via their respective memory-allocation/deallocation routines. This greater dependence upon the SMM routines increases the possibility that the user will be unaware of their presence and will neglect to load correctly.

So what's wrong?

New facilities, such as stack-based code and multiprocessing, are now available and are intimately dependent upon the SMM routines. These newer facilities were not considered when the original decision was made to let the user's memory-management package control free memory. Hence, the SMM and user packages were not designed and implemented to execute correctly in an environment of concurrent execution. Each requires protection around its treatment of the heap. Luckily this type of protection is easy to add by using the multitasking LOCKON and LOCKOFF primitives.^{4,5}

Likewise, the MZGET mechanism for coexistence of SMM and the user memory manger also causes problems. The current scheme can lead to unrecoverable errors. The two most likely errors are due to recursive execution through the locked portions of the SMM and stack-overflow manager (\$\$STKOFEN). Let's look at both of these errors.

The first problem is brought about by calling a subroutine that causes a stack overflow. When the overflow is detected, \$\$STKOFEN is invoked. \$\$STKOFEN runs the SMM routines to allocate another stack segment so that execution may continue. It uses a private stack for this purpose, and while running SMM it locks access to the private stack. However, suppose that while the private stack is locked an SMM routine overflows the private stack and again calls \$\$STKOFEN. At this point we end up in a deadlock, waiting for access to the private stack.

The second problem is very similar to the first. This problem is caused by a call to an SMM routine overflowing its stack while the heap structure is locked. In this case,

\$STKOFEN is invoked and it locks access to the private stack on which it runs the SMM routines to allocate a new stack segment. However, since the heap structure is locked, this allocation request will wait—resulting in another deadlock.

The primary cause for these problems is the presence of the MZGET routine. In the absence of this routine there are methods which would allow us to remove the problems. That is, the SMM routines could be rewritten to not allow a stack overflow to occur while the heap structure is locked. However, as long as the MZGET routine exists and may be contained in the user's memory manager, we can make no such guarantee.

I should mention that these problems can currently happen under NLTSS whether or not the application code is stack-based or multiprocessing. The fact that the majority of all support libraries for NLTSS are stack-based is sufficient. As an aside, these problems are also possible under LTSS if an application (knowingly or not) makes use of stack-based code.

What can we do?

A new version of SMM has already been written to address the new facilities and remove the problems mentioned above. We hope to replace the current NLTSS version with this in the near future, although for some transition period it is likely that the current LTSS-like memory manager will also be available. The proposal for memory management under NLTSS can be summarized as follows:

- SMM will provide a library of basic memory-management primitives. The desire here is to develop SMM into a set of primitives upon which higher-level memory managers may be built. Most of the basic functionality already exists in SMM. It may, however, become necessary to add new primitives to support needed functionality for higher-level memory managers or for the memory-management needs of new languages. This should also make it easier to port outside applications, using memory management, to our facility.
- All other memory-management packages will be required to be modified to reside on top of the SMM. This will put SMM always in control of the free memory space. The desire to have the user's memory-management package control free memory is understandable. However, it is felt that, within the new environment, the many diverse memory-management usages (stack-based, multiprocessing, and the large SMM usage by utilities and libraries) relinquish any real control the user had. (A new MMLIB has already been converted to reside on top of SMM).
- The hierarchical design required by the previous condition obviates the need for the MZGET bridge routine. Therefore, the routine will not be contained in the new SMM.
- Ad hoc memory management will be disallowed. The ad hoc scheme must be modified to reside on top of SMM (as previously described). In order to disallow ad hoc memory management, all support library routines that allow users to explicitly expand their program field length will be removed (e.g., IZSCMLCM). Remember, any use of a routine like IZSCMLCM constitutes a form of memory management. In this proposal the only way in which two or more memory managers can exist in the same application is by basing them all on the SMM routines.

For further information on the proposed memory management scheme or the problems described, please contact me (L-300, MAIL name `res11`) or Bob Cooper (L-16, MAIL name `bcooper`).

Heap positioning can make a difference!

The following problem is a concern for both LTSS and NLTS applications loaded for the Cray X-MP/48.

For purposes of this description, let's assume we are using LDR to load a segmented application. (A nonsegmented application is a special case of this because it can be considered to have only a root segment.) When LDR detects usage of the SMM routines it sets aside in the application an initial free space to be controlled by SMM. This space is known as the *initial heap*. By default, LDR places this initial heap just after GOBCOM in low memory and before the code and common blocks that make up the root segment (see Fig. 1). This is an appropriate place for the heap because the root segment is always present and therefore so is the heap. LDR also supplies directives to control the position and size of the heap.

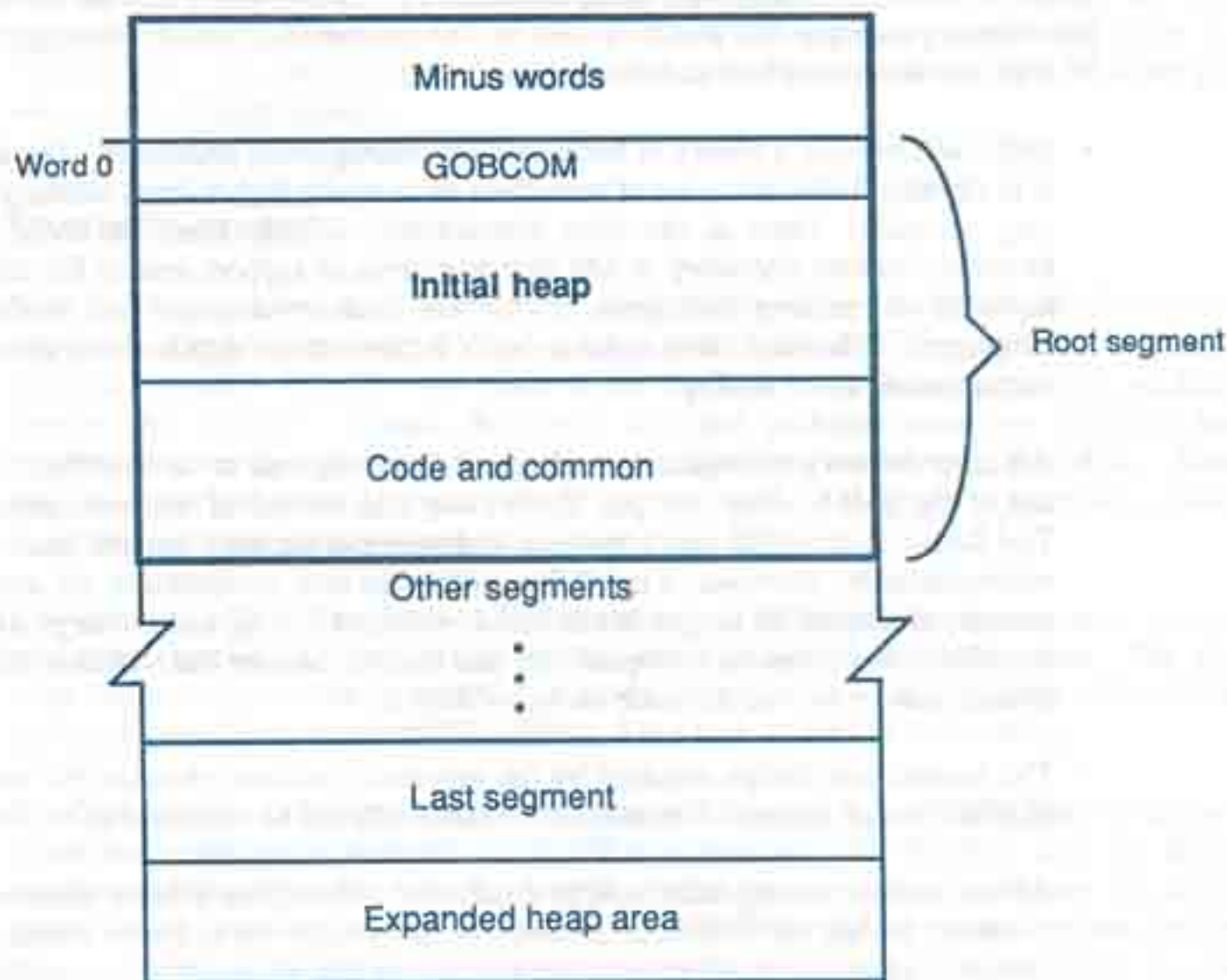


Fig. 1. In loading a segmented application, LDR positions the initial heap in the root segment, after GOBCOM and before the code and common blocks.

LDR loads a segmented application with an initial field length that includes only the root segment. As the other segments are needed, the field length is automatically increased until the longest segment chain is in memory. This allows a code to execute at a reduced size until the larger size is actually needed.

Consider an allocation request to the SMM. If the request cannot be fulfilled from the initial heap, SMM will expand the field length to gain more free space. In the case of a segmented code, the expanded area must be placed beyond the last segment. SMM expands the field length to encompass the longest segment chain and the needed area. From this point onwards SMM manages a discontinuous heap.

Now, with the groundwork laid, let's get down to the heart of the problem. If the user specifies the size of the initial heap (via the LDR directive) in order to avoid a discontinuous heap, it is possible to cause the memory size, starting from the base of the root segment and ending at the top of the last segment, to exceed 4 million decimal words. Cray X-MP/48 hardware limitations require executable code to exist below this memory barrier. Expanding the initial heap may cause executable code in the last segment to be pushed beyond the 4-million-word mark.

We can avoid this problem by removing the initial heap from the root segment and placing it just after the last segment in the application. However, this has the effect of canceling LDR's ability to start the segmented application with a reduced field length. The application would end up with a field length large enough for all its segments and the initial heap very early in its execution.

I would like to get feedback on whether the capability for initially reduced field length (and the SQUEEZE directive) is vitally important to applications at LCC. We are trying to address this problem but we need more input from the user community.

For further information on this problem, please contact me (L-300, MAIL name `res11`) or Mary Zosel (L-300, MAIL name `zosel`).

References

1. B. Atkinson and G. Sutherland, "System Memory-Manager," *Tentacle* 3 (11), 21-27 (November 1983).
2. R. Cooper, *Users Manual for MMLIB—A General-Purpose Memory Manager*, Lawrence Livermore National Laboratory, Livermore, CA, report LCSD-416 (23 September 1983).
3. Work performed by Rick Johnson, LLNL.
4. Gary Blair, "Synchronization Mechanisms and Partitioning Programs," *Tentacle* 6 (10), 3-11 (November 1986).
5. *Multitasking User Guide*, Cray Research, Inc., Mendota Heights, MN, Technical Note SN-0222 (March 1986).
6. Rick Johnson, *LDR*, Lawrence Livermore National Laboratory, Livermore, CA, report LCSD-344 Rev. 3 (23 October 1984).

Robert E. Strout II is a computer scientist in the Language Group of User Systems Division (USD).

CMRD Reports**MATHLIB vs NMATHLIB***Tokihiko Suyehiro*

Some confusion has arisen in the use of the public mathematical library MATHLIB because we have two different operating systems (LTSS and NLTSS) and different hardware (Cray-1 and Cray-X/MP) at the Livermore Computer Center. The purpose of this article is to clarify the different MATHLIB versions and the systems they run on.

The version of the library compiled with LINK=1 (nonreentrant) is called MATHLIB. Public file MATHLIB is the same on all the Cray-1 computers and the Cray-X/MP. It runs under the LTSS system. The version compiled with LINK=2 (reentrant) is called NMATHLIB, and it is available under the LTSS system on all Cray-1 computers and on the H machine. *Caution:* Because some instructions are different for the Cray-1 and the Cray-X/MP computers, a code loaded with NMATHLIB on the H machine will not run on a Cray-1.

The only NMATHLIB available under the NLTSS system is on the B machine. This version has software locks in the routines that process error messages and hardware locks in the random number generator. (See Mark Durst's article, "Random-Number Generators for Multiprocessing" in the May 1986 issue of *Tentacle* for an explanation of the need for the latter.)

The following table clarifies the relationship between the math libraries and the systems they run on:

Hardware type	Machine name	System	LINK=1 library	LINK=2 library
Cray-1	C,D,E,G	LTSS	MATHLIB	NMATHLIB
Cray X-MP	H	LTSS	MATHLIB	NMATHLIB
Cray X-MP	B	NLTSS	(none)	NMATHLIB

Because NMATHLIB (LINK=2) has no dependent libraries, it can be loaded with either LTSS or NLTSS Fortran libraries (i.e., FORTLIB or NFRTLIB, OMNILIB or SOMNILIB, etc.). MATHLIB (LINK=1), on the other hand, has OMNILIB and FORTLIB as dependent libraries.

Tokihiko Suyehiro is a computer scientist in the Library Software Group, Computing and Mathematics Research Division (CMRD).

LATEST FROM LABNET

George Pavel

Service goals

A basic service describes how a client physically attaches to LabNet so that bits can be transferred. An enhanced service provides a useful function on top of a basic service.

LabNet basic services

- Ethernet Bridge:* We will make Ethernets in different buildings look like a single network.
- Lab-Wide Ethernet:* We will connect isolated computers in different buildings with an Ethernet that spans the Lawrence Livermore National Laboratory (LLNL).
- Dedicated Line:* We will provide, where possible, other types of high-speed data communications between buildings. This service is negotiated with each client individually.

LabNet enhanced services

- Octopus Access:* We will provide file transfer and terminal emulation to the Livermore Computer Center (LCC) computers from systems on Closed LabNet.
- ARPANET Access:* We will provide access to the ARPANET/MILNET network from any system on Open LabNet using the TCP/IP communications protocol.

Achievements

The backbone cable systems for Closed and Open LabNet have been improved for better reliability. Both systems have new head-ends that will simplify testing of the backbone.

A new version of the LINCOS file transfer utility for Sun workstations on Closed LabNet provides performance numbers that indicate once the data starts moving between a Sun and a Cray, the throughput is about 500,000 bits per second.

The Closed LabNet backbone and routers have maintained greater than 99% uptime on a 24-hour-a-day operating schedule. Nearly all interruptions in service were scheduled updates to the routing table which take just a few minutes.

The gateway between Open LabNet and the ARPANET/MILNET is finally in operation after months of effort. Our first attempt suffered from a variety of software problems that were never solved. We scrapped that design since by that time we were able to purchase a commercial unit to do the job, a unit from Proteon that is now in production and appears to be stable. The Computation Department's unclassified Unix computer (LLL-LCC) uses this gateway as its main link to the ARPANET/MILNET. Any computer on Open LabNet that has software implementing the TCP/IP protocols can make use of this gateway; contact Andy Beals (Ext. 3-8595) for details on how to do this.

Current Work

We will perform a complete sweep of the Open LabNet cable system to balance the network that has grown dramatically since a sweep was last done. This work is expected to take at least one month and will cause disruption along the network at times. Once the system is balanced, the addition of new connections will no longer cause problems for existing connections.

The connection of Buildings 131, 321, and 381 to Closed LabNet is imminent. The only work remaining is the mounting of the enclosures for the Applitek routers within the buildings. The connections to Buildings 131 and 321 will mark the first appearance of VMS Vax computers on Closed LabNet outside the Computation Department.

LabNet Connection Summary — November 17, 1986

Open LabNet

<u>Active</u>		<u>Pending</u>	
<u>Building</u>	<u>Program</u>	<u>Building</u>	<u>Program</u>
117, 116	Computation	113	Computation
121	EE-iCAEn	194	Nuclear Chemistry
131	EE-iCAEn	2103	Computation
141, 1450, 1477	EE-iCAEn	235	Chemistry
1478, 1403	Earth Sciences	281	Nuclear Chemistry
1405, 1413			
1477			
151	Nuclear Chemistry		
177, 175, 179	AVLIS		
3724, 3725,			
3726, 482, 490			
218, 216, 2106	Computation		
2684	EE-SDEG		
311	Physics		
315	Finance		
316, 219, 319	Computation		
451	NMFECC		
551W	AD Computation		

Closed LabNet

Active		Pending	
Building	Program	Building	Program
111	Nuclear Design	131	ME
111	R	321A	MFD
113, 116, 2103, 2106	Computation	381	LASNEX
		5475	FEL

George Pavel is manager of the LabNet Project. He can be reached through electronic mail as gp@111-crg or gp@111-lcc or ICDC::PAVEL or on the Crays with MAIL name gp.

PC CORNER

Rich Serbin

Is your Macintosh or IBM PC a security violation?

Recently several people have phoned me asking questions about "hooking up" their Mac to the Octopus computer system. They wanted to use the Mac as a terminal emulator. What I discovered was that they had not submitted a computer security plan with the appropriate Computer Security Coordinator. If you are using the Mac, or any other computer as a terminal emulator on the Octopus System, you **must** file a Security Plan.

Let me state the actual policy: "It is the policy of the Laboratory that no classified work can be processed on any computer terminal or system until a computer security plan has been approved by the LLNL Computer Security Organization."

The policy also states that, "A short form (one page) security plan must be submitted for all microcomputer-based systems used for classified or sensitive information. A short form also is required for any microcomputer (classified or unclassified) connected to a modem or to any network."

After you have filled out the form(s) and your terminal has been approved for processing classified information, an authorization sticker will be affixed to your terminal.

If you need security plan forms or assistance, please contact these people:

- Microcomputer security plans—Lonnie Moore, Ext. 3-9273.
- Networks—George Bush, Ext. 2-1463.
- Mainframe systems—Ray DeSaussure, Ext. 2-4238.

Forms (LL 5491) are also available from forms and records.

Storing classified data on disk

A couple reminders for use of the Octopus System. Remember that Octopus is a classified system! Even if your work on it is unclassified, you must protect the system and your work to the PARD level as a minimum. This means all output must be marked PARD if you can't immediately inspect it and verify its actual classification. The Octopus system, including your terminal, must also be *Protected As Restricted Data*, and be locked when unattended at night. Any lock (Z, administrative, key on the IBM-PC/AT, etc) is acceptable. A PC used as an Octopus terminal that has a hard disk must be protected as PARD and locked up at night.

A more serious problem exists when you "capture" classified data to a disk. If the "capture" is to a floppy disk, you must secure the floppy in a repository or approved vault. But if the "capture" is to a hard disk, you must secure the computer; if you are using a removable disk cartridge, secure the disk cartridge in a repository or vault. For this reason a Mac with a non-removable hard disk is **not** recommended for use on Octopus.

A warning about Univation's Slim Line Removable Disk Unit

Last month I evaluated Univation's Slim Line two 10-Mb removable cartridge disk drives. At the time of the evaluation I had several types of disk failures, and a cartridge developed bad sectors. While I was experiencing these failures, the Advance Warning System (AWS1 and AWS2) did not work.

After I published the article, I unplugged the unit from my PC and removed the cover from the Slim Line unit to see why the Advance Warning System wouldn't work. What I discovered was that the Advance Warning System would never work. It was connected directly to the power supply! The only warning possible from this connection was a power-supply failure, not a disk-drive failure! The unit does not function as advertised.

I suggest that you do not use the AWS1 and AWS2 as an indicator of disk problems, since they don't work. To verify that your data is still "good" you run CHKDSK /F (which is on your DOS disk) once in a while. Another utility you can use is the Norton Utility DT, which not only flags disk problems, but also corrects certain disk defects.

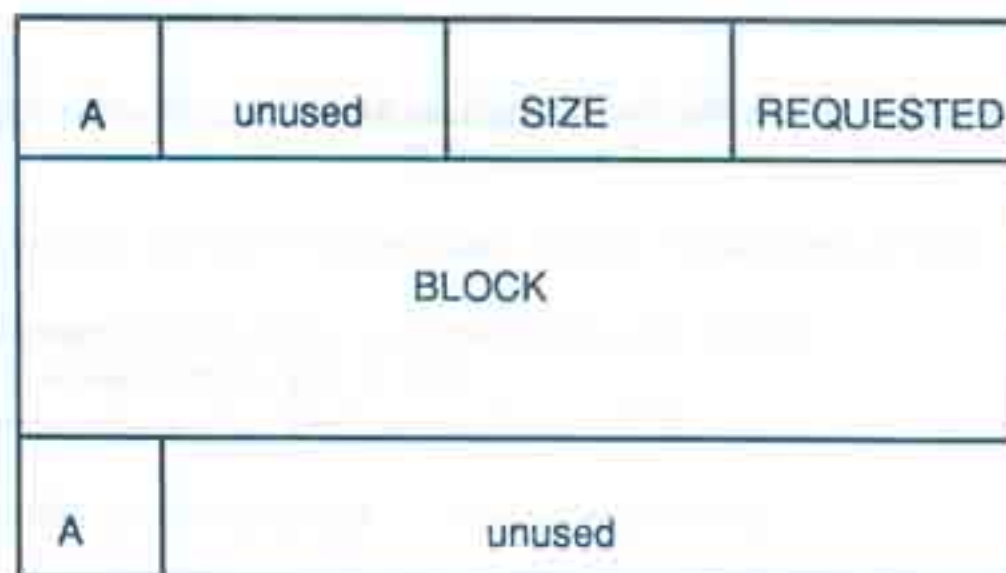
Rich Serbin is a computer scientist in the Applications Systems Division (ASD) and PC Support Team Leader.

Consulting Office Commentary**VULNERABILITY OF LTSS SMM CONTROL WORDS***Maurie Louis*

The control words that SMM (System Memory Manager) uses to manage the memory in the heap are vulnerable to user indexing errors. In fact, an array that is overwritten in the heap could very easily destroy an SMM control word. I'm going to describe how the control words are arranged by SMM and give an example of how one of them could easily be lost.

As I wrote in my October 1986 *Tentacle* article "Introduction to LTSS SMM," the heap is an area of memory set up by LDR for use by the SMM. Blocks of memory within the heap can be allocated or deallocated using the FORTLIB routines GETSPACE and RELSPACE or by using the MZ routines (see "What's Behind GETSPACE and RELSPACE," in the November 1986 *Tentacle*). Each block of memory in the heap, whether it is allocated or free, contains control words.

An allocated block has two control words, one at the beginning and one at the end:



A Tag field
 1 bit
 (1 = allocated; the tag fields will be equal in the top
 and bottom control words of the same block)

SIZE Size of this block
 24 bits

REQUESTED Size requested when this block was allocated
 24 bits

The unused bits in the control words may contain garbage because of previous allocations and deallocations. This causes no problem, however, because the bits are unused.

A free block of memory has three control words associated with it, two at the top of the block and one at the bottom:

A	unused	SIZE	FORWARD
unused			BACKWARD
BLOCK			
A	unused	SIZE	unused

A	Tag field 1 bit (0=free)
SIZE	Size of this block 24 bits
FORWARD	Pointer to the next free block in the linked list 24 bits
BACKWARD	Pointer to the previous free block in the linked list 24 bits

These control words establish a linked list of free blocks that is not ordered with respect to memory address. The free block pointed to may not be the closest by address.

The initial heap is located after the low-core words of memory (unless LDR received an HLOC option). If the heap is extended, the extended part of the heap is placed immediately following the initial field-length. After this occurs, there are two 'pieces' of heap: the initial piece and the extended piece. Boundary control words define the boundaries of these pieces of the heap and are located at the beginning and the end of each piece:

A	unused			
HEAP				
A	B	unused	FORWARD	unused
NON-HEAP				
A	unused			
HEAP				
A	B	unused	FORWARD	unused

A Tag field
1 bit
(always = 1)

B Btag field
1 bit
(Always = 1; when displaying in octal there will be the octal representation of the bit string 1nn where nn are the two high order bits of the leftmost unused field)

FORWARD Pointer to next piece of the heap
24 bits
(0000 = no more pieces following)

How to Trace Through a Heap

Starting at the second word of the named common block NCMEMCOM is a two-word block of memory which I'll call the HEADER block. This HEADER block connects the ends of the linked list of free blocks and makes a circular linked list. All that is contained in the block are its two control words that are structured like the two first control words of a free block. The SIZE field of the first control word contains zeros meaning it's a zero-length free block. Its FORWARD field contains the address of the first free block in the linked list and its BACKWARD field contains the address of the last free block in the

linked list. As we'll see in my example, the BACKWARD field of the first free block points to this HEADER block and the FORWARD field of the last free block points to the HEADER block. Again, this block is contained in the named common block NCMEMCOM.

Following is a sample trace through a heap that has allocated and deallocated blocks. In allocating the blocks, SMM had to extend the heap so there are two pieces.

1. Use TRIX to look at the LDR map and determine the start of the heap.

```
i=bin,x=xsample,ml=map,mo=full
```

```
cray loader - 111c 06/25/86
```

```
time and date of load is - 09:18:20 10/23/86 , machine is c
program length is 00014027
transfrer address = 00012076a
memory management heap is at 0000201, size of 00011610
```

2. Use DDT to locate NCMEMCOM and display the HEADER block.

```
ddt +sample / 1 1
DDT version 9.4e June 13, 1986

loc ncmemcom
00015330pa = NCMEMCOM:NCMEMCOM

di 155331b for 2
00015331 = 000000000000000000000202
00015332 = 0000000000000000000027171
```

3. Display the first word of the heap, as reported by the LDR load map, which is the first boundary control word.

```
di 201b
00000201 = 100000000000000000000000
```

4. Display the second word of the heap which is the first control word of the first block.

```
202b
00000202 = 000000000000120000007042
```

5. Calculate wherethe next block begins by adding the current address to the size field of the first control word.

```
print 202b + 120b
0000000000000000000000322
```

6. Display the calculated address less 1 which is the bottom control word of the current block.

```
321b
00000321 = 000000000000120000000000
```

7. Display the calculated address which is the top of the next block.

```
322b
00000322 = 100000000002116000002116
```

8. Again calculate where the next block begins.

```
print 322b + 2116b
0000000000000000000002440
```

9. Display the calculated address less 1, the bottom control word of the current block.

```
2437b
00002437 = 10000000000000000000000000
```

10. Display the first control word of the next block.

```
2440b
00002440 = 100000000001606000001606
```

Example of Tracing Through a Heap

The following list includes all the sets of control words, including the boundry control words, of an entire heap. It shows how you can trace a heap by using the SIZE fields or how you can determine the linked list of free blocks by following the FORWARD and BACKWARD pointers.

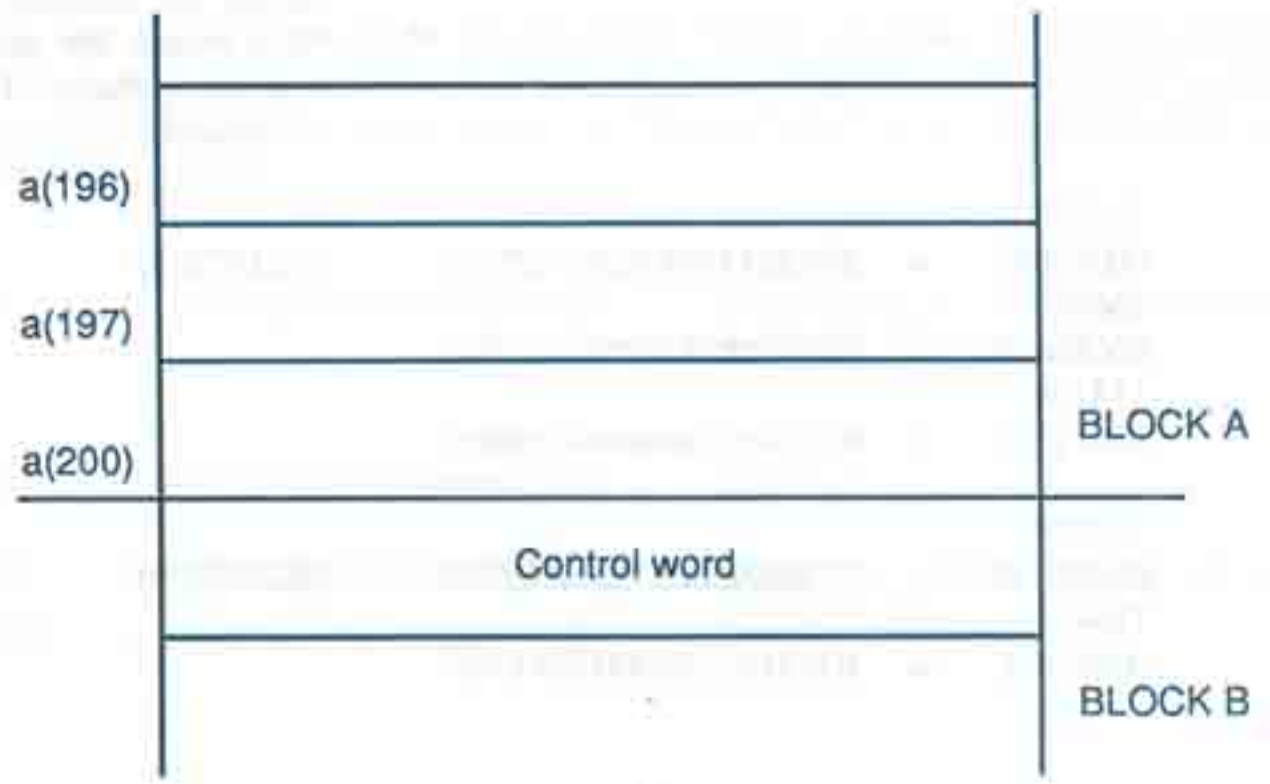
```
201b
00000201 = 10000000000000000000000000    (Top boundry control word for the
                                                    first piece of the heap.)

202b
00000202 = 0000000000012000007042    (BLOCK 1)
203b
00000203 = 00000000000000000015331    (Note the 15331 pointer to the
321b                                           HEADER block.)
00000321 = 0000000000012000000000
```

322b			
0000322	=	10000000021160002116	(BLOCK 2)
2437b			
00002437	=	10000000000000000000	
2440b			
00002440	=	1000000000160600001606	(BLOCK 3)
4245b			
00004245	=	10000000000000000000	
4246b			
00004246	=	1000000000144200001442	(BLOCK 4)
5707b			
00005707	=	10000000000000000000	
5710b			
00005710	=	1000000000113200001132	(BLOCK 5)
7041b			
00007041	=	10000000000000000000	
7042b			
00007042	=	0000000000076600010652	(BLOCK 6)
7043b			
00007043	=	000000000000000000202	
10027b			
00010027	=	0000000000076600000000	
10030b			
00010030	=	1000000000062200000622	(BLOCK 7)
10651b			
00010651	=	135261002351240150000	(Note that these words are not cleared before the relevant fields are filled. This means the unused fields may contain garbage. This is perfectly acceptable.)
10652b			
00010652	=	0000000000045600027171	(BLOCK 8)
10653b			
00010653	=	0000000000000000007042	
11327b			
00011327	=	0000000000045600000000	
11330b			
00011330	=	1000000000031200000312	(BLOCK 9)
11641b			
00011641	=	1000000000000000000000	

11642b			
00011642	=	1000000000014600000146	(BLOCK 10)
12007b			
00012007	=	1000000000000000000000	
12010b			
00012010	=	1400000002717000000000	(This is the bottom boundry control word of the first piece of the heap.)
27170b			
00027170	=	1000000000000000000000	(This is the top boundry control word of the second piece of the heap.)
27171b			
00027171	=	0000000002734200015331	(BLOCK 11)
27172b			(Again note the 15331 pointer to the HEADER block.)
00027172	=	000000000000000000010652	
56532b			
00056532	=	0000000002734200000000	
56533b			
00056533	=	1400000000000000000000	(This is the bottom boundry control word of the second piece of the heap. The forward pointer is zero so this is the end.)

Now for an example of how one of these control words can be overwritten. If you allocate a 200 word array, A, that is located at the end of an allocated block in the heap and erroneously set A(201) to 0, it will destroy a SMM control word. Usually, the program continues running for a while before crashing. If an allocation or a release occurs after the control word is destroyed, a high probability exists that the program will terminate with error #204.



Now that you have read this article, you can trace through the heap to find the erroneous control word. The next step is to backtrack to the point where the original error occurred. You can do this by running the program in iterations using DDT and checking the control word that eventually gets overwritten. The SAVE option in DDT is very helpful in this situation.

Maurie Louis is a computer scientist in the Systems Consulting Group, User Systems Division (USD).

SELECTED OCTOGRAMS

(These originally appeared in past daily Octograms. They are reprinted here because of their topical interest.)

Problem with XPORT: STAGE command—October 10, 1986

Yesterday we discovered that when files are staged to MASS from the ATL, their directory information is altered such that they become MASS ONLY. When the MASS copy expires, the file is lost. This problem started in August.

To avoid lost files, please check your entire directory of files and ARCHIVE those that are marked FC when they should be marked FL.

```
xport lst . lev. n
```

where n is the number of levels in your largest directory.

If a MASS file has expired and the tape copy is lost, report your user number and the file chainname to the Coordination Center. The file will be restored in your directory as soon as possible.

I apologize for this inconvenience.

Micheal Nemanic, Storage/NSD
Ext. 3-7726

USERINFO update October 14—October 13, 1986

Tuesday morning, October 14, from approximately 6 AM until 8 AM, the USERINFO system will be unavailable while it is being updated and tested with new versions of USERINFO, USERUP, and USER4UP on all Crays. This update will implement the removal of the existing terminal-to-TMDS monitor table, and will add a TMDS monitor field to each user record. A payroll account field will also be added to each user record. Users who own codes that access the existing USERINFO file's TMDS table should have made changes to those codes to adjust to the new file format. (If not, read Cray text file `.381338:userinfo:changes` for conversion information.) The NEW versions of such codes should be installed by their owners as soon as possible AFTER the USERINFO update has been made. Users should expect errors, or unexpected prompts for TMDS number from such codes, during the period after the update has been made, but before the codes have been updated by their owners. An attempt has been made to fill the TMDS field in each new user record with a correct TMDS monitor number (using a data base assembled by the Coordination Center for other purposes), but there may be cases of this number being incorrect or having a zero value (meaning no TMDS number is known for this user). If this situation occurs, the affected user should run USERUP to check and possibly set the TMDS field correctly, using the following commands:

```

userup / t v
list
set tmds=mmmm
list
update
used!]
read
end

```

[lists current values]
[set correct monitor number]
[see pending changes]
[note: update TMDS is no longer
used!]
[the changed record]
[if happy]

(Note: if you need to do this, why not take the opportunity to correct any other fields in your record that are not correct?) Announcements of the status of the update will be made by TMDS message. Questions about any of this can be directed to:

Terry Heidelberg, SIG/USD
Ext. 2-4154
Doris Ryon, SIG/USD
Ext. 2-4345
Len Lyles, SIG/USD
Ext. 2-3773

New FILES for H machine—October 21, 1986

A new version of FILES will be placed in public on the H machine. This new version allows a new option `ssd` to display extra information about SSD files. Shadow files have the text (shadow) after the file name, scratch SSD files have (SSD) after the name.

The `ssd` option requires the large file index to be read. This causes the FILES utility to expand its field length in most cases when the `ssd` option is used. Therefore, it has been decided that the option will NOT become the default for the utility, since interactivity would suffer as a result.

Rob Haynes, SIG/USD
Ext. 2-7202

Cray FORTLIB—October 27, 1986

On Monday, October 27, FORTLIB version 18b will be placed in public on the D, E, G, and H machines. This is the same version that has been on the C machine for the past week. This new release supports ANSI 77 I/O features for CIVIC-compiled codes including direct access files, internal files, list-directed I/O, and the OPEN, CLOSE, and INQUIRE statements. You can obtain more details about the extended I/O features by using XPORT to read `.446450:ans77doc`. The August 1986 *Tentacle* also contains an article describing the new features.

This update of FORTLIB also fixes several minor bugs related to the use of implicit unit specifiers, namelist I/O, and format widths for character I/O

If you have questions or problems with this update, please contact either one of us at the extensions listed below.

Arlene Getchell, SIG/USD
Ext. 3-6349
Donna Derby, CRI
Ext. 3-2602

100 pt/in file format on RJET—October 29, 1986

The RJET 300pt/in laser plotter can now emulate a 100pt/in Versatec much as the 200pt/in Versatec can emulate the 100pt/in Versatec. The old 100pt/in format should now work properly on the laser plotter.

Bob Cooley, NSD
Ext. 2-4395

CC: Public file update—October 30, 1986

The public file CCOMP (the C compiler used by CC) will be updated Tuesday, November 4. This C compiler is considerably faster and has had several bugs fixed.

The public file CC will also be updated with a new C controller program. Documentation on the execute line is available as follows:

```
exe newcc ccdoc x.  
allout rjet nn ccdoc box ann ccdoc
```

The new controller and compiler are available now by using the public file NEWCC.

Kelly O'Hair, LG/UDS
Ext. 2-4296

TEXSPOOL updated at LCC—November 3, 1986

TEXSPOOL in .150888:tex has been updated to version 4.0. The new version allows output to the TMDS. For example

```
texspool test.dvi tmds=116
```

L. Chase, SSD
Ext. 2-4086

ALLOUT update on the C machine—November 3, 1986

ALLOUT was updated on the C machine to fix a bug that occurred if 33 files were submitted for output. This ALLOUT will be moved to all other Cray machines in a week.

Gail Whitten, SCG/USD
Ext. 2-3704

H machine file index—November 6, 1986

Due to the file index problems we have been experiencing on the H machine, it has been decided to increase the size of the file index on the XMP to 42,074 (dec.) on the 18th of November at 07:30.

Anyone who has a routine which looks at the file index on the XMP will need an updated version for the XMP.

Walter D. Headley, NCC
Ext. 2-4531

LCC documentation queue—November 6, 1986

The LCC document queue for November is now online in DOCUMENT. To view the queue, log onto any Cray computer and enter

```
document view queue entire
```

The routine will request that you press the return key, type a number, or type `all`. The carriage return will yield a screen of printed matter on your monitor, a number will yield the number of lines you request, and `all` will yield the whole document.

If you wish a printed copy of the queue, enter

```
print queue hsp box ann id
```

where *ann id* is your output box number and identification.

Lila Abrahamson, TID/Computation
Ext. 2-5680

Via Desk-Top Publishing

PRODUCING THE *TENTACLE*

Elsa Pressentin

The *Tentacle* has been produced offline via desk-top publishing since last August. (Our last regular online issue came out in June, and in July we published the 1985 Index, also an online effort.) With four offline issues behind us, we are ready to discuss our adventures in the world of desk-top publishing. I will also explain our production procedures and comment briefly on some of the software and hardware that we use in our effort.

Software and hardware considerations

We use a Macintosh Plus computer, a hard disk, and a LaserWriter Plus. Efficient use of the Mac demands a hard disk. Although we do not have one yet, I also recommend an external drive; it is easier to manipulate files. The software we use each month to produce the *Tentacle* includes MacWrite, Microsoft Word, MacDraw, MacΣqn, and PageMaker. MacWrite and Microsoft Word are both word-processor software. We use MacWrite in the conversion process only. We prefer Microsoft Word for editing and formatting.

We use MacDraw for line drawings and flow charts. MacΣqn is a desk accessory that provides an environment for writing technical equations into Macintosh word-processing programs (see "MacΣqn for Setting Equations" on page 37). PageMaker is a desk-top publishing software tool or electronic page make-up system.

Microsoft Word vs MacWrite

Why do we use Microsoft Word instead of MacWrite for the "heavy-duty" word-processing work? For several reasons, actually, but primarily because PageMaker accepts Word documents better than it accepts Write documents. The ruler, on which line lengths and tab settings are based, is not as accurate in Write. When you transfer a Word document to PageMaker, most of the tabs and margins remain intact. However, the tabs and margins in a Write document have to be re-adjusted in PageMaker, which is not an easy task. (PageMaker does not like major edit changes and blows up when you attempt to add large chunks of text, justify lines, or adjust margins.) We also prefer Word because it will accept Write documents whereas Write will not accept Word documents. Word is, in my opinion, a more sophisticated word processor and less troublesome when used with PageMaker. A new, even more powerful version of Word is due out next month.

Submitting articles for publication in *Tentacle*

We accept articles for publication four different ways: Macintosh diskette, IBM PC diskette, online (via LCC's Octopus system), and hard copy.

Since we produce the *Tentacle* on a Macintosh, receiving an article on a Macintosh diskette is of course our preference. When you bring your diskette to us, we can begin to

format and edit it immediately, especially if your article has been prepared with Microsoft Word. If however, you prepare an article using MacWrite, it is fairly simple to convert it to a Word file!

When we made the change to desk-top publishing, we gave serious thought to authors who prepare and submit articles online via Octopus and those who prepare and submit IBM PC articles. How could we capture those keystrokes? And what about our own editorial staff who do not have Macintoshes? The solution was obvious. We needed some type of transfer or conversion process. With a combination of equipment (see related article on how we transfer online and IBM PC articles to the Macintosh on page 38), we have succeeded. All we ask is that your article be an ASCII file.

So, if you plan to submit an article for publication that you prepared online, write your ASCII file as follows:

```
xport!write .903904:tentacle:yourfile!end
```

If you prepare your article on an IBM PC, bring a copy of your article and the diskette (ASCII file please), and we will convert it.

Our fourth option is to submit only a hard or paper copy of your article. Bring your article to us before (preferable) or by the first working day of the month, and we will key the article during the editing phase.

Editing and author review stage

Since we prefer Microsoft Word, most articles are converted to that word-processing program for editing. We set the line length, headline style, font, etc. at the same time your article is edited. After your article has been edited, you will be contacted for a review. Usually at the author review stage your article will be a Microsoft Word-produced document, with all the current formatting in place (headline style, line length, font, etc.). Because some of our staff members do not have Macintoshes, however, it is possible that your review copy will look nothing like what it will in its final version. If you wish to see a Microsoft Word or PageMaker version of your article for final approval, simply let us know. Because we have encountered some problems in PageMaker, we have established a final review in which typographical errors can be corrected, margin errors fixed, or technical/data changes made.

Once the article has been edited, reviewed, and corrected, it is ready to be placed in PageMaker. We set up "master" pages for each section of the *Tentacle*. The proper headers (section title) and footers (date, name of publication, and page numbers) are a part of the master page. We are using version 1.2 which still has a lot of bugs. We are anxiously awaiting version 2.0, due out this month.

Life with PageMaker

Aldus Corporation's PageMaker was the first desk-top publishing software on the market. To achieve that, I believe that they sacrificed a lot. On the surface PageMaker seems impressive. But, a user who has publications and/or typesetting background, and thus higher (more professional) expectations and standards, might be disappointed. Like *PC News* (see their article on desk-top publishing in their September/October issue), we encountered pinnacles and pitfalls in the changeover to desk-top publishing.

First, what is PageMaker and how does it work? PageMaker (PM) is a desk-top publishing software tool. By using PM, you can combine text and graphics on a page and produce a publication. First you create documents (files) with other Macintosh applications such as MacWrite, Microsoft Word, and MacDraw. Once those documents are ready, you "place" them into PM. You can move the text and graphics around in PM until you achieve the look you want: one-column (full page), multi-column format, etc.

If it sounds easy, it is! A demonstration of PM is quite impressive. But (yes the proverbial BUT), using PM can be frustrating because of the bugs, unclear error messages, and poor documentation (to name just a few of my complaints). It is difficult to edit in PM, even though the manual says you can. Changing a tab or trying to justify text often results in catastrophe (the file blows up and you lose all your work). Other weaknesses that PM has include no kerning (controlling space between letters), no way to automatically create an index, unadjustable leading (space between lines of text). PM increases leading, for example, when a line of text contains super- or subscripts, resulting in noticeable leading changes.

Tables are another frustration in PM. (And a quick glance through this issue will tell you we had a frustrating month!) Although most tab settings come over from Microsoft Word accurately, a few do not. Sometimes it's an easy fix; merely adding a backspace or tab lines everything up again. But often, PM simply blows up. Another problem I have encountered in PM occurs when I attempt to "place" drawings on a page. Sometimes the outer edge of the figure is missing! One solution is to draw an invisible box (use the white pen) around your figure while you are still in that application.

Last stages in the production cycle

After the issue is finally complete, we print out final or master pages on the LaserWriter Plus. Then we go into a formal review cycle (Computation Department, User Systems Division, staff, patent, classification, Director's Office). We print out a final copy and take it to the printer. In our case we take our camera-ready copy to Mel Moura, TID's printing coordinator, and the *Tentacle* is sent offsite to be printed. Printing takes seven working days. When the printed version arrives at the Mail Room, we go over and check the issue and release it for distribution.

Conclusion

Although the overall tone of this article may seem a little negative, we are still excited about producing the *Tentacle* using desk-top publishing. Yes, there are bugs still to be worked out. And we are anxiously awaiting PM's new version. Let me add here, that Aldus Corporation is very helpful; if the software representative you get can't answer your questions when you call, they do call you back! But all in all, we are able to produce a higher quality, more professional looking publication in less time.

Elsa Pressentin is an editor/writer in the Technical Information Department (TID) supporting the Computation Department in User Systems Division (USD).

MACΣQN FOR SETTING EQUATIONS

Jane Winter

MacΣqn is helpful for setting equations that can be pasted into documents produced with Macintosh word-processing programs. The documentation is good. For my very first attempt at setting equations, I read through the documentation once. Then I was able to set all the equations on pages 33 through 35 of the August 1986 *Tentacle* by referring back to the appropriate sections.

MacΣqn is a stack-based graphics editor. A line along the bottom of the screen shows what font you are using, the point size, the menu item you are working on, and the level you are at in the stack.

Limited editing now possible

Very little editing or correcting is possible with MacΣqn. Version 1.7, recently released, does allow some very simple editing. Now you can replace a character or a sequence of characters. You can also insert or delete a character or a sequence of characters. MacΣqn makes a simple attempt to adjust the other characters to accomodate the insertion or deletion. Versions before 1.7 required you to redo the entire equation from the point of insertion, deletion, or correction.

Using MacΣqn

I use MacΣqn with MicroSoft Word. I start editing as usual. When I come to an equation, I switch to MacΣqn, leaving MicroSoft Word still active. I set the equation, copy it, switch to the MicroSoft Word window (without closing MacΣqn), and paste the equation in its proper position. I then print the page to see whether the equation looks like I thought it would look. Unfortunately, the screen image of the equation does not give a very accurate picture of what spacing will look like when the equation is printed, and you cannot make any changes to the equation after it has been pasted into your document. To correct spacing you have to go back to the original equation in the MacΣqn window where you can move one character at a time one point size at a time to the left, right, up, or down.

I prefer to set one equation at a time and paste it into my document immediately. It is possible, however, to set all of your equations, paste them into your scrapbook, and then paste them in their proper positions. Each MacΣqn window must be placed as a complete unit. You cannot set two or three equations in one window and then place them in different parts of your document; nor can you place an equation on the same line with other text.

I find using MacΣqn quite tedious. But, I'm sure it is much easier than specifying each change in point size, placing each character individually, and drawing in your special characters.

Jane Winter is a computer support technician in the Technical Information Department (TID) supporting computer documentation in the User Systems Division (USD).

GETTING FILES FROM THE PC TO THE MAC

Jane Winter

Articles for the *Tentacle* come to us in a variety of ways. Some are sent online; a few are delivered on Macintosh diskettes; more are supplied on IBM PC diskettes; and some arrive as only hard copy. When we switched to desk-top publishing, we had to find a way to transfer articles to the Macintosh from all these sources.

Those arriving on Macintosh diskettes were the easiest to handle, of course. Those arriving as hard copy have to be completely typeset, anyway. Not wanting to rekey the articles that come to us online or on PC diskettes, we searched for a way to transfer them to the Macintosh.

We decided on a system that uses an IBM PC with an LTERM terminal emulator and VersaTerm as a terminal emulator for the Macintosh. Articles must arrive in the form of ASCII files for the following procedure to be successful.

Moving a file

If the article arrives online, we first download it to the PC. From the PC we send the file to the Pyramid computer. To do this we plug the PC into our modem and dial the number for the Pyramid. After logging in, we upload the file from the PC to the Pyramid and print it on the screen to be sure it is readable.

Next we unplug the PC, plug the Macintosh into the modem, open VersaTerm on the Macintosh, and dial the Pyramid. We log in and then use SAVE STREAM to capture the file on the Macintosh as we print it from the Pyramid.

Converting to MacWrite and Microsoft Word

After quitting VersaTerm, we open the file with MacWrite and examine it to be sure that everything transferred correctly. Although we use Microsoft Word for the *Tentacle*, we first convert each article to a MacWrite document because MacWrite allows us to eliminate all end-of-line carriage returns when the file is converted. If we converted articles directly to Microsoft Word documents, we would have to manually delete each end-of-line carriage return. We specify margins, tabs, fonts, etc. and do the final editing after the document has been converted from MacWrite to Microsoft Word.

Jane Winter is a computer support technician in the Technical Information Department (TID) supporting computer documentation in the User Systems Division (USD).

PUZZLES

Henry Larson

Integer cryptoglyph

In the cryptoglyph below, each of the symbols *A, B, C, D, E, F, G, H, I, J, K, L, M, N* stands (in one-to-one correspondence) either for one of the digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 or for one of the operators +, -, *, /. Each six-symbol block is an arithmetic expression having an integer value between 0 and 27. What is the hidden message?

DBJMNE LIGDML NFLGFJ NAFAIH JMIHAJ

CEGFAH NBFAID ICNGKE

IHBNME DJMEBD DAIKMJ DEGJBD.

Solution to October's *Digital Counterfeit Detection*

The detection can be made with only three weighings. Moreover, the exact weight of a good coin and of the counterfeit can be determined. The method is as follows:

Label the coins *a, b, c, d, e, f*. For the first weighing, weigh *a, b, c,* and *d* together; this will give a value W_1 . For the second weighing, weigh *c, d,* and *e* together; this will give a value W_2 .

If $3W_1 = 4W_2$, the fake is *f*; the weight of a true coin is $W_2/3$; and a third weighing, of *f* alone, will determine the weight of the counterfeit.

If $3W_1 \neq 4W_2$, then for the third weighing weigh *a* and *c* together; this will give a value W_3 . The complete story is then determined in the following manner:

If ...	Then a true coin weighs	The fake is	The fake weighs
$W_1 = 2W_3$	$W_3 / 2$	<i>e</i>	$W_2 - W_3$
$2W_2 = 3W_3$	$W_2 / 3$	<i>b</i>	$W_1 - W_2$
$W_1 + W_3 = 2W_2$	$W_2 - W_3$	<i>c</i>	$2W_3 - W_2$
$2(W_1 - W_2) = W_3$	$W_1 - W_2$	<i>d</i>	$W_2 - W_3$
$3(W_1 - W_3) = 2W_2$	$W_2 / 3$	<i>a</i>	$W_1 - W_2$

Obviously, in each case we can tell whether the fake is heavy or light.

Correct, on-time solutions were received from Frank Barish, Ralph Hager, John Fletcher, Joanne Levatin, Ken Zahonas, H. Flocard, David Kershaw and Garret Boer. Levatin reports that an extension of this method finds the fake among 18 coins in 5 weighings.

Solution to November's *Balanced Integers*

Part A. Tabulating the first few integers, we find that 10, 12, and 14 are balanced from 1. Since at most one prime occurs for each two numbers greater than 14, and since 15 and 16 are not prime, no larger such integer can occur.

Part B. In order to be unbalanced, an integer n must not be balanced from any other integer. This implies that n must be unbalanced from, at least, its nearest "potential" balancers, $(n - 3)$ and $(n + 3)$. The smallest integer meeting this last criterion is 23. By an argument similar to that of Part A, it is clear that 23 cannot be balanced from any larger integer, and testing of all smaller integers reveals that there are none which balance from it.

Correct, on-time solutions to at least one part were received from John Fletcher, Bill Moran, Frank Barish, Ralph Hager, Pat Weidhaas, and Rich Belles.

Editor's Note

If you wish to contribute solutions or comments, send them to PUZZLES, L-73. Responses received before the next *Tentacle* deadline can affect Henry Larson's solution, scheduled to appear here next month.

Computist's Corner

GEOMETRICAL ATOMS

R. K. Cralle

Although the triangle is inherently stronger for construction work than the square, we rarely use it, except in geodesic dome construction. Because a square (Fig. 1) can be more easily pushed over (as in Fig. 2) than a triangle, in the construction of houses, builders brace the corners with diagonally placed pieces of wood. This, in effect, adds the strength of triangles. The contribution of *Aushaus* (Fig. 3) didn't help much, since his method didn't even add first order improvement to rigidity (Fig. 4). Some of you may have experienced this kind of structural failing—at night, in high winds—if you are old enough and have lived in remote areas.



Fig. 1



Fig. 2



Fig. 3



Fig. 4

Consider the following geometrical problem: how can you brace a square (with unit length sides) using only an arbitrary number of said unit length pieces? The only connection permitted is at the corners with pins or bolts allowing two pieces to pivot. In your quest, if you are lucky or fortunate enough to have an Apple Macintosh™ computer, you can, using MacDraft™ (not MacDraw™, which doesn't allow rotations other than 90°), avail yourself of a homemade graphics erector set (all of the figures in this article were so made).

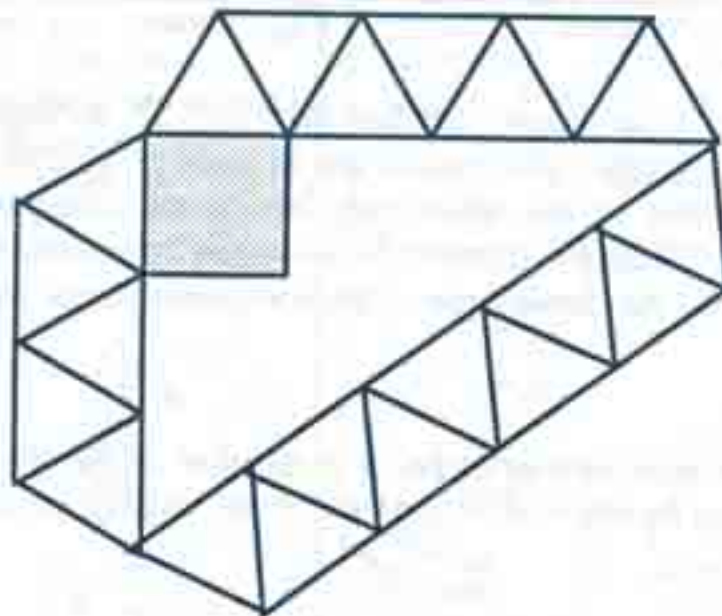


Fig. 5

Figure 5 is one solution to the problem. The resulting triangles are all equilateral (since that is all you can build given only unit-sized pieces). The method used involves building a triangle (the smallest integer one, with a right angle, with sides of 3,4,5) around the square. The constructed sides are rigid, plaited girders as seen on many bridges. When these sides are connected to the square's vertices, the entire structure is as rigid as one of the single triangles. This, then, is a sort of geometrical atom. The set of sticks (sides) combined into this structure becomes what they, in a pile, were not. This is what hydrogen atoms do for a living—they form all of the rest of the atoms and molecules of nature. At least this was true before we had four score or so fundamental particles. (The neutron was discovered the year I was born. "That long ago, eh!") By itself there is no free, uncontained hydrogen—its escape velocity allows it to leave the Earth.

Lest you get carried away with the rectitude of your computer-driven erector set, Fig. 6 depicts a seeming solution to the proposed problem. That is, vertices *a* and *b* appear to be one unit apart and, thus, could be connected by one of the rigid-triangles structures. A moment's calculation reveals, however, that the actual distance is ~ 1.07 —an error sufficient to warrant the invention of molding (as actually did happen in house construction). You could build Fig. 6 (and no one would be the wiser), and triangles would keep the square rigid—all you would need to do is to make a slight error in drilling the bolt holes (a fudge not permitted in geometry). House construction (sometimes incorrectly called home construction—you can't build an abstraction with wood) couldn't survive without small errors.

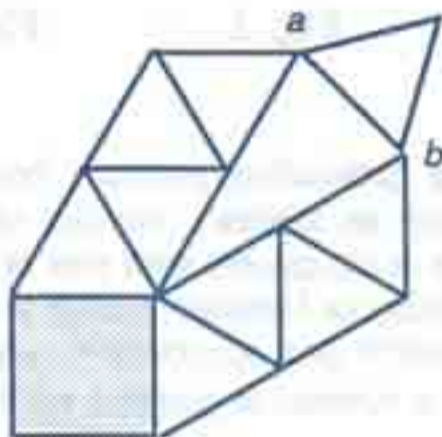


Fig. 6

Is there a better (less pieces) solution than Fig. 5? In searching for one, remember that because of the triangles, dimensions are in units of $\sqrt{3}/2$ ($= \cos 60^\circ$) in one direction, and half unit segments in the other (two equilateral triangles making a rhombus have a diagonal $= \sqrt{3}$). Thus, the distance between any two vertices in your construction must be comensurate with the dimensions of the composite pieces you connect.

R. K. Cralle, venerable computist-at-large, is a member of the Workstation Group, User Systems Division (USD), and may be reached through electronic mail at cralle@lll-crg.ARPA.

THE OCTOPUS'S GARDEN

Joanne Perra and Neale Smith



(Continued on next page)

FOUR UP TO THE HILL

I'LL DRESS UP ZODD, MY HOUSEHOLD ROBOT AS JOLLY OLD ST. NICK TO GREET DIVISION MEMBERS AND COLLECT THE MONEY.



I'LL HAVE TO HAND IT TO JOE THIS TIME. THIS IS GREAT! NONE OF HIS USUAL FOURUPS. MAYBE 2 MONTHS AT BRILEY JR. HIGH CHANGED HIM.



THIS HOLIDAY AMBIENCE IS WONDERFUL! REMINDS ME OF CHRISTMASSES LONG AGO. MY FAVORITE HOLIDAY MEMORIES INCLUDE 3 THINGS: GOOD FRIENDS, A COZY FIRE, AND A MUG OF EGGNOG.



DOES IT SEEM WARM IN HERE?



ALL WE NEED NOW IS THE EGG NOG.

Joanne Perra is a computer scientist in the Systems Consulting Group, User Systems Division (USD). Neale Smith is a computer scientist in the Workstation Group, User Systems Division (USD).

Reader Input Form

From:

Name _____

Group/Company _____

Mail Stop/Address _____

Phone _____

My comments are

Building Research Forum

Fold Here

The Tentacle
Computation Department
University of California
Lawrence Livermore National Laboratory
Mail Stop L-301, Box 808
Livermore, CA 94550

Fold Here

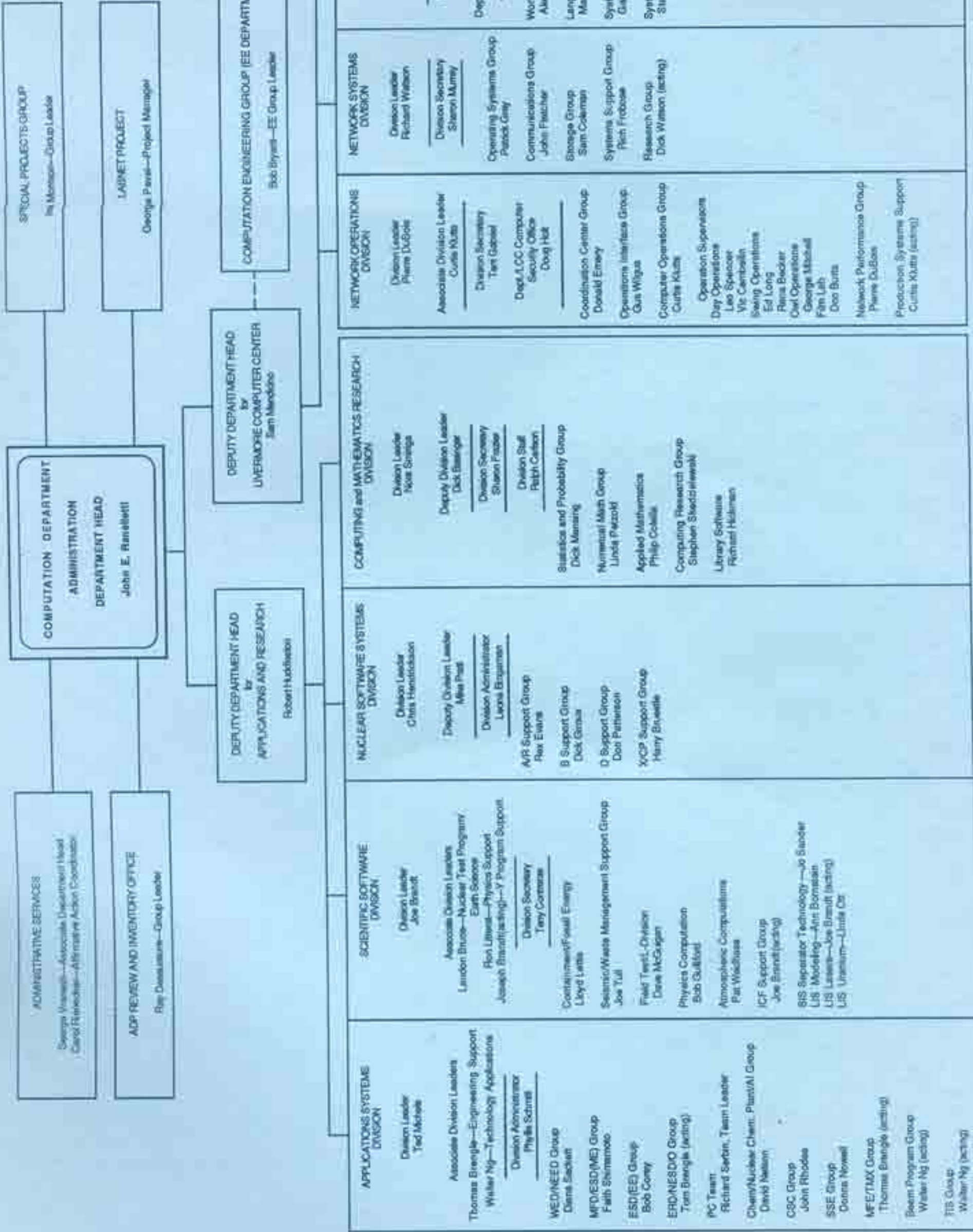
This document was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness or usefulness of any information, apparatus, product or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial products, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government, and shall not be used for advertising or product endorsement purposes.

Work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract number W-7405-Eng-48.

Copyright 1986 the Regents of the University of California. All Rights Reserved.

ORGANIZATION OF THE COMPUTATION DEPARTMENT

December 1966



Computation Department
University of California
Lawrence Livermore National Laboratory
Mail Stop L-301, Box 808
Livermore, CA 94550

