An Interactive Software Engineering Tool for Memory Management
and User Program Evaluation

Submitted for consideration to the
National Computer Conference
1974

Wolfgang W. Millbrané
Juan Rodriguez-Rosell

Brown University
Center for Computer and Information Sciences
Providence, Rhode Island 02912
Telephone: 401-861-4821

This work is sponsored in part by the National Science
Foundation, grant GJ-28401X, the Office of Naval Research,
contract N00014-67-A-0191-0023, and the Brown University Division
of Applied Mathematics.
TABLE OF CONTENTS

Abstract
I. Introduction..............................................................1
II. System Organization......................................................3
III. Parameters of Evaluation of Program Memory Utilization........7
IV. Some Case Studies........................................................11
    FRESS - A File Editing and Retrieval System.....................11
    The CSS Editor............................................................12
    SCRIPT and NSCRIPT.......................................................13
    Fortran F Compiler.......................................................13
    Some CPU Simulation Programs.........................................14
V. Some Practical Guidelines to Increase Memory Utilization........15
VI. Conclusions and Future Work..........................................17

References...........................................................................20
ABSTRACT

As the use of virtual memory becomes more and more accepted, the problem of effective storage management becomes more and more important. To date most efforts to optimize the use of memory have been directed at devising memory management strategies at the operating system level that minimize the number of page faults. Little effort has been made to provide the programmer with suitable tools for making his programs "more local". To fill this need the Brown University Display for Working Set References was developed, enabling the programmer to directly monitor the page referencing behavior of his modules. This interactive graphics measurement system utilizes a satellite processor to display user program memory references and working set parameters. The system and several parameters used to evaluate program memory utilization are discussed. Practical programming guidelines and packaging techniques to improve memory usage in a virtual memory environment are presented.

Keywords:
Virtual Memory, Program Packaging, Software Measurement, Program Behavior, Satellite Processors
I. INTRODUCTION

As the use of virtual memory becomes more and more accepted, the problem of effective storage management becomes more and more important. To date most efforts to optimize the use of memory have been directed at devising memory management strategies at the operating system level that minimize the number of page faults. For example, Comeau [1] has shown that the loading sequence of subroutines can have a considerable effect on paging activity. Hence page fault-optimizing loaders, linkage editors and compilers have been proposed. Although the concepts of "locality" and "working set" have been known for some time (c.f. Denning [4]), little effort has been made to provide the programmer with suitable tools for making his programs "more local". This seems to stem from the fact that, short of notions of "modular coding", little is known about what sorts of programming habits actually result in local code. Consequently, most optimization techniques used to date have assumed that user programs were an unmodifiable input to the operating system.

Techniques for increasing locality of user programs and thereby reducing the paging overhead in a virtual memory environment have been investigated by Comeau [1], Hatfield [7] and Ferrari [5]. The methods presented involve the automatic restructuring of program modules into relocatable segments to increase the likelihood that page references that are close in
time will also be close in space. In the Hatfield study the use of online displays to determine optimal segment sizes and view the effects of program reordering was found to be exceedingly helpful, but use of their system as an everyday programmer feedback and user program monitoring tool was never fully explicated.

The Brown University Display for Working Set References (affectionately known as BUDWSR) was primarily developed as a user oriented tool to fill the need for user feedback systems by enabling the programmer to interactively monitor the memory referencing behavior of his modules. It was hoped that the programmer would be able to get a "feeling" of what it means to write localized code, and hence be able to modify his programming techniques in order to reduce, or at least have more control over, the memory resources required by his program. Although BUDWSR has not been in existence long enough for us to evaluate its effectiveness as a programmer training or feedback tool, we have already been able to establish some simple guidelines that allow the programmer to measure and hence increase the memory utilization of his programs. These guidelines could be used as a basis for establishing engineering standards for program evaluation. Furthermore, BUDWSR has proved itself to be an extremely powerful systems programmer tool that greatly facilitates the manual repackaging of modules in order to increase memory utilization.
II. SYSTEM ORGANIZATION

The system (c.f. Figure 1) essentially consists of a System/360 machine language interpreter and a satellite display processor. The user runs his program in much the same way as he would in a normal CP/CMS [2] environment except that the interpreter counts the references made to memory pages (the page size is user defined) and periodically transfers its tables to the satellite computer. The interpreter runs in a 512K byte virtual machine and simulates a 256K CMS environment for the user's program; the data gathered thus reflects only the activity of the user's program and is not dependent on the virtual machine's external environment. Since the System/360 instruction set is highly formatted, the basic data gathering facilities of the interpreter are fairly simple and represent an acceptable cost (roughly 50 times the cost of the original program) for gathering memory utilization statistics. The basic display produced by the satellite consists of page addresses plotted against time (measured in numbers of instructions). For each page address and time interval the system plots a small vertical

---

*Most RR format instructions can be executed directly via the System/360's execute instruction, while most RX, RS, and SI format instructions can simply be executed after computation of the base displacement address. Currently, BUDWSR does not handle complete interrupt processing, e.g., interpretation stops when an interrupt is generated by the instruction stream, and resumes when control is returned from the interrupt handler. Since many of the CMS nucleus pages are in shared core, references to the nucleus code should not result in any significant paging overhead to the user.*

-3-
spike whose height indicates how often that page was referenced during that time interval (see trace of the Fortran compiler in Section IV). At the start of the trace the user may define an arbitrary page size (from 8 bytes to 128K bytes) and time interval (from 1 to 65,535 instructions). Furthermore, he may ask the satellite to perform a non-destructive compression of the reference data, e.g., the user may ask the interpreter to gather statistics for 128 byte pages, and then view the results for 128, 256, ..., or 128K byte pages. This allows him to view the programs behavior both on a global level (using for example a 4K byte page size) and to selectively "zoom-in" on any local peculiarities. To aid the programmer in identifying and restructuring inefficient modules, a cumulative reference count for each page, and module names and entry points (obtained from the CMS loader tables) are also displayed.

Figure 1: BUDWSR Block Diagram.
The satellite part of the system also uses the page reference data to produce a graph of the working set size versus time. Thus the user not only has available which specific memory pages were referenced but also a summary of the total memory used during a time interval. The working set size is computed in bytes for the user specified page size and also for the System/360-67 page size (4K). With a small page size (typically 128 bytes) the difference in the working set sizes indicates the amount of wasted space in the System/360-67 working set. Currently, we have measured only working set sizes and re-entry rates for an individual process. In reality, the amount of memory allocated to a process is a function of the paging algorithm and the load of the system. (Some data on system page fault and re-entry rates can be found in Rodriguez-Rosell and Dupuy [9].) We have not concerned ourselves with simulations to determine page fault rates as a function of memory size or the operating system's memory management policy. In general the amount of real memory available to the user is beyond his control; however, by using BUDWSR, the programmer can improve the locality of his programs while developing them, conceivably even changing to algorithms or data structures that might prove less noxious to the system. In return, he hopes that if his process uses memory efficiently the operating system will allow his process to execute rapidly.
The use of the satellite processor [10] came about quite naturally in that we approached program monitoring as two separate tasks, data gathering and analysis. We also wished to be able to monitor the output of the tracing program while it was running, rather than wait for offline hard copy output of the execution record of the monitored program. Thus the satellite performs the multiple tasks of communication with the System/360 interpreter, updating the disk-resident master file containing the page references, and updating the display itself as new data are transferred by the interpreter or when the user "scrolls" through, or locally alters the page size for the memory display. The use of the satellite to process the trace data makes the system extremely flexible. Thus whenever we wish to determine whether or not a specific memory utilization parameter is meaningful, it is a simple matter to add code to the satellite program in order to calculate and display the parameter. This type of flexibility will become even more important when the data gathering facilities are incorporated into the hardware of the machine running the process to be monitored.

Another effect of using a satellite processor rather than a process running on the virtual machine itself is the fact that the interpretation and display data handling proceed in parallel in "real time." The interpreter runs unmolested by user

---

2In general, the information to be displayed is larger than the available display area (e.g., the CRT display can effectively represent 50 4K-byte pages, but not 800 256-byte pages).
interrupts for larger page sizes or different display lists. Thus it is possible to view a trace of a system, repackage it, and/or retrace it with different parameters all in the span of a few minutes. For monitoring an interactive system\(^3\), we can view a trace at the same time we are specifying commands to that system in order to quickly isolate and reduce the memory requirements for inefficient command modules. Since the trace data are immediately available after each time interval, it is also unnecessary to correlate the data from an entire run (such as would be given by an offline plot) to the specific command or sequence of commands that generated the data. It is of course also possible to view previously taken trace data in a standalone mode of the satellite.

III. PARAMETERS OF EVALUATION OF PROGRAM MEMORY UTILIZATION

To date the system has demonstrated its usefulness through several practical applications. Many of the facts noticed about program behavior could of course have been predicted by use of common sense, and the reader may thus find some of the results rather un-startling. However, the fact that most of the observed traits are found in a large class of programs indicates that common sense rules (such as presented in the next section) are

\(^3\)For example, some of this paper was edited under a text editor being monitored (see below).