ON COST OF STATIC LINKING AND LOADING OF SUBPROGRAMS

Norman Beck, Gordon Ashby
University of Oregon

INTRODUCTION

The purpose of this paper is to report some data concerning cost in CPU processing due to loading programs. The data was collected on a PDP-10, using modifications made by the linking loader to the prologue generated for FORTRAN compiled programs, by the addition of one UUO (a programmed operation similar to an SVC on IBM 360/370), and several cells in the monitor used as counters. The data covers the number of programs loaded and the CPU ms expended loading them. This data is broken down between programs that were loaded and never entered and programs loaded and eventually executed. It is further classified according to periods of heavy use for program development and periods of heavy production use.

The following paragraphs will present the reason behind the collection of the data, describe the modifications made in order to collect the data, summarize the data, describe the University PDP-10 configuration and general work load, and finally remark on the significance of the findings.

BACKGROUND

For some time the authors have believed that static linking and loading was inefficient and a significant contributor to system overhead. Examples of dynamic linking and loading functions are found in systems that also incorporate sophisticated memory management, auxiliary storage management, and scheduling functions, e.g., MULTICS. However, most medium and large scale systems *in use* employ static linking and loading and less sophisticated memory management and scheduling. We are attempting to discover some use patterns of subprograms incorporated into static load modules and also something of the costs and desirability associated with forming static load modules. This paper reports aspects of the study conducted thus far.

One of our contentions has been that static loading, such as is done on the PDP-10 and IBM 360, results in the linking and loading of numerous programs which never enter execution because of early termination due to errors and because of the dictates of the data portion of particular runs. We have suggested that dynamic loading techniques could be the default function for an operating system in order to free the internal memory and ease the CPU and channel overhead involved in linking and loading. Because of overhead involved with dynamic loading, we have further suggested that static loaders should be available to the programmer who has developed a production program so that he/she could combine those portions of the load module which would standardly be needed in a run of the program. Finally, the output from the static loader should be acceptable input to the dynamic load function of the system.

In order to initially test the validity of our contentions, we decided to collect performance data directed at the following three questions. Are a very large portion of the programs linked and loaded never entered? Is the CPU cost of loading them significant? Do production runs differ sufficiently from developmental runs to warrant inclusion of static loaders in the operating system? This report describes a study based on these three questions. Again, the study is directed at a problem of non-paged systems and does not apply to paged memory management systems.

THE MODIFICATION TO FORTRAN OBJECT PROGRAMS AND THE LOADER

As mentioned earlier, the modifications to the system to collect the data amounted to 1) having the linking loader displace one word of the prologue for FORTRAN object modules with a UUO call that points to an argument block, 2) a UUO monitor level service routine, and 3) four counters in the monitor named %UOLTL, %UOLTT, %UOLUL, and %UOLUT.

There are two arguments to the UUO monitor call. The first is the displaced instruction and the second contains a loading time in ms in the left half and a count in the right half. For main routines, the loading time is the CPU time expended on all subprograms linked into the load module and the count is the number of subprograms linked. For subprograms, the load time is the CPU time expended on that module and the count is set to zero.

The UUO level routine upon entry checks the count for non-zero to see if the entry is to a main routine. For main routines the count field is added to the %UOLTL and the loading time field is added to %UOLTT. Thus %UOLTL and %UOLTT at any given time contain the total number of routines linked and loaded and the total CPU time in milliseconds used for linking and loading programs. For a subprogram, %UOLUL is incremented by one and the loading time field is added to %UOLUT. %UOLUL represents the subset of the %UOLTL routines that have been entered and %UOLUL the CPU time expended linking and loading the subset. The UUO level routine replaces the displaced instruction in both cases so that subsequent entries do not result in duplication of the logged information.

On the PDP-10 a user can *save* a core image for

repeated execution immediately after linking and loading. Our modifications were designed to capture duplications that result from multiple executions of these saved files. We chose to count all subroutines loaded and used at every execution of the FORTRAN main program. The overhead involved in linking and loading saved files that are never executed is not included in the data. The bias introduced by this choice should tend to increase the percent used of those loaded and therefore tend to the apparent desirability of dynamic linking/loading facilities. Again, the data portion for a particular execution of a program is a determining factor in which subprograms are executed and the above philosophy was required in order to allow this variable to function.

The counters are written to disk at regular intervals in records that were date and time stamped. A simple program then categorizes the data according to shift, day, and production or developmental period. Totals and subtotals for programs loaded, loaded and used, loaded and unused, and corresponding loading times are easily available. A summary of this information, collected over a four-month period from January through April 1973, follows.

SUMMARY OF DATA COLLECTED

Table 1 gives the totals collected for each of the four months January through April 1973, and composite four-month totals. Over the four-month period approximately 3.23 hours of CPU time was spent linking and loading 121,172 FORTRAN subprograms. Of these, 65,541 were never entered, wasting approximately 1.83 hours of CPU time. During the four-month interval used to collect the data, the PDP-10 was operational for 1,531 hours and CPU utilization was approximately 300 hours. Thus, a very small portion of available CPU time was used performing linking and loading functions, 3.23/1531 = .0021 or .21% with the unused figure being .12%. Whether loading time is compared with available CPU time or used CPU time, it doesn't appear to consume a significant amount of the CPU. On the average, it takes approximately 100 m/s of CPU time to load a program.

Another of our questions was with regard to the portion of programs that were loaded and went unused for one reason or another. Less than half, 45.91% of the programs loaded into memory were used in our four-month sample. (Frankly, we were surprised that the percentage used was this high.) These programs occupy core memory whenever the program is running or ready to run. While we cannot tell the amount or percentage of internal memory space so used, we note that, on the average, programs that were not entered took more CPU time to link and load than did programs that were entered, 101 m/s versus 91 m/s, and we can surmise that a significant amount of internal storage in our computer configuration supports unneeded programs and data.

	January	February	March	April	Total
Total Loaded	20,645	31,658	21,402	47,467	121,172
Total CPU Time Loading	1,947,860 m/s	4,109,903 m/s	2,305,749 m/s	3,322,064 m/s	11,685,576 m/s
Average CPU Time	94 m/s	130 m/s	108 m/s	70 m/s	96 m/s
Total Used	8,717	13,667	11,754	21,493	55,631
Total CPU Time for Used	838,264 m/s	1,439,501 m/s	1,055,993 m/s	1,737,533 m/s	5,071,291 m/s
Average CPU Time	96 m/s	105 m/s	90 m/s	81 m/s	91 m/s
Total Unused	11,928	17,991	9,648	25,974	65,541
Total CPU Time for Unused	1,109,596 m/s	2,670,402 m/s	1,249,756 m/s	1,584,531 m/s	6,614,285 m/s
Average CPU Time	93 m/s	148 m/s	130 m/s	61 m/s	101 m/s
% Used of Those Loaded	42.22%	43.17%	54.92%	45.28%	45.91%

Table 1

Table 2 presents the four-month totals for our normal student and developmental shifts Sunday through Saturday as opposed to our limited access "night owl" production shifts. (A brief profile of our system appears in the next section.) While the sample from production shifts is smaller than that for the normal shifts, there does appear to be a significant difference in the portion of routines loaded and used, 69.73% for production times and 43.99% for normal times. (Again, we were surprised that the difference wasn't larger.) Even in the case of production times, we conjecture that significant internal memory space is wasted on programs and data that are never referenced. Again we note that the average CPU time used in loading unused programs exceeds that expended on programs that are entered, 121 m/s versus 101 m/s.

Figure 1 shows a plot of % used of programs loaded for the four individual months collected. The difference in percentage entered was clear for each month collected. Again, the difference was not as pronounced as we had anticipated it might be.

	students & developmental	primarily production
Total Loaded	112,152	9,020
Total CPU Time Loading	10,719,923 m/s	965,653 m/s
Average CPU Time	96 m/s	107 m/s
Total Used	49,341	6,290
Total CPU Time for Used	4,436,337 m/s	634,954 m/s
Average CPU Time	90 m/s	101 m/s
Total Unused	62,811	2,730
Total CPU Time for Unused	6,283,586 m/s	330,699 m/s
Average CPU Time	100 m/s	121 m/s
% Used of Those Loaded	43.99%	69.73%

Table 2

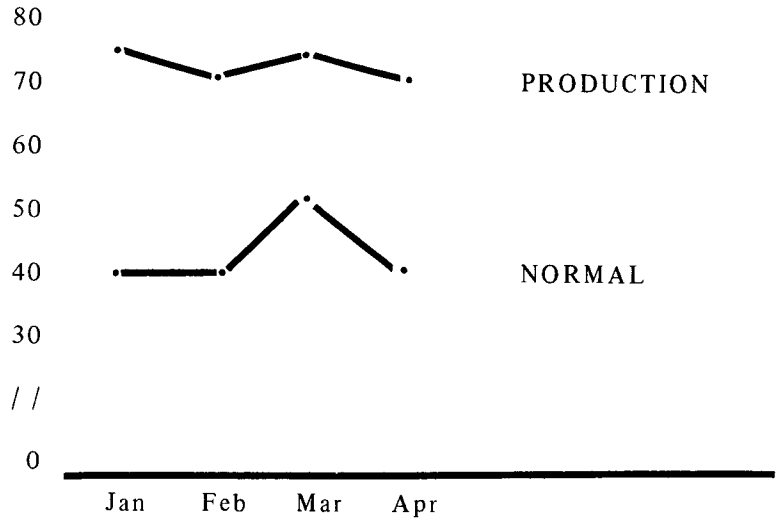

Figure 1

BRIEF PROFILE OF U OF O PDP-10 SYSTEM

The PDP-10 at the University of Oregon is used entirely as an interactive timesharing system to support research and instruction. During the time the data was collected, the system was configured with 96K of internal core memory and the monitor was housed in approximately 36K. During normal shifts the individual user is restricted to a core maximum of 26K and is charged for terminal connect time, CPU time used, and core memory used. The policies tend to cause the system to be used for developmental purposes during these shifts, and response time is generally instantaneous for the user.

During night production shifts, night owl, the core maximum restriction was raised to 40K words and the charge for core memory used was dropped. Additionally, the teletype room in the computing center was closed so that use of the system was restricted to research users on campus with terminals at their sites. Larger production jobs were run in these time periods but not to the total exclusion of editing and developmental runs.

The major language subsystems included in the PDP-10 system are FORTRAN, COBOL, ALGOL, BASIC, and MACRO-10 (an assembler). Of these, BASIC is interpretive and not subject to the linking loader, COBOL had not been installed at the initiation of the data collection and has yet to receive significant use, and MACRO-10 and ALGOL are not used much by either students or research users. Thus, of the two subsystems that represent the large majority of the use of the PDP-10 system, BASIC and FORTRAN, FORTRAN was appropriate for collection of data concerning the linking and loading function.

CONCLUSIONS

Linking and loading does not appear to use significant CPU time. On the PDP-10 approximately 100 m̸s is expended, on the average, in linking and loading a routine. Ironically, we observed a higher average time for linking routines not entered than for linking routines that were entered.

We observed that less than half of the programs linked and loaded were used during some point in the run. A significant amount of memory probably was occupied by programs and data regions that were never called for during the run. These programs and data would never enter the address space of the job or process if dynamic linking and loading techniques were employed. Systems need not employ elaborate paging and segmentation schemes and share programs and data to benefit from dynamic linking and loading techniques. However, better memory management than that exhibited on OS/360 is needed to realize some of the advantages. While we felt that the percentage of never-referenced routines would be higher than it appears in our data, we still believe that standard use of dynamic linking and loading facilities would be preferable to using static ones.

A difference is apparent between production and developmental runs. Our data showed that between 20 to 30% more of the routines that were loaded were in fact entered. Intelligent use of a static loader to combine those routines that are very likely to be used in a run of a program could save a production user money over time, the cost of 100 m̸s of CPU time for each instance of the linking and loading of a routine on our system if it employed dynamic techniques. This could amount to a substantial amount for a single application to be run repeatedly over time and a static loader appears to be warranted in a system which also employs dynamic techniques.

Linking and loading affects the functioning of systems in numerous ways not examined by this study. It places demands on the file handling portions of the system, upon the logical and physical I/O portions of the system, and upon scheduling functions. We hope to examine these aspects of linking and loading in the future.

REFERENCE

DEC SYSTEM10 Assembly Language Handbook, Digital Equipment Corporation, 1972.