To: Distribution

From: R. A. Barnes

Date: 03/30/79

Subject: The cost of call-push-return

For some time now there has been some controversy on the subject of call-push-return overhead in Multics and whether something should be done to reduce it. Usually the figure of 5% is quoted as the system overhead for call-push-return, which has in the past led many to conclude that nothing should be done about it. However, I have maintained that for many subsystems on Multics and for many user applications this figure should be increased. Using the modern disciplines of structured programming in constructing subsystems leads to the use of many small procedures. Because recursive algorithms often provide the most natural solution to problems, and because a subsystem is easier to maintain if its procedures are separately compilable, it is often undesirable to avoid the call-push-return overhead by the use of quick procedures. For example, we avoided the call-push-return overhead in the new fortran compiler by using only seven external procedures with hundreds of quick procedures. However, it takes 180 seconds of CPU time to compile one of these 8000-line procedures, and it's much more difficult to divide work on the compiler among more than one programmer. Because of the above, it seemed important to estimate the call-push-return overhead for a subsystem with many small external procedures.

As an example of a subsystem with many separately compiled procedures, I chose the PL/I compiler. First, it was necessary to derive the cost of the average call made by the compiler. In the cost of a call I include argument list preparation, stack frame push and pop, and return. As an educated guess, I assumed that the average call had five arguments without descriptors. Using the vclock builtin around null-loops, loops making quick calls, and loops making non-quick calls, I got the following results:

	quick call	non-quick call	
in cache	6 usec	70 usec	user code
out of cache	8 usec	76 usec	

(This test case assumes that the pl1_operators_ sequence remained in the cache.)

The PL/I compiler was then used (with all links snapped) to compile its own optimizer which cost 40 seconds of virtual CPU time. Finally trace -auto on was used to count all external calls made by the compiler while compiling the optimizer. This came to 94110 external calls. My knowledge of the workings of the compiler leads me to believe that at least 6000 more non-quick internal calls are made, so I assume that the compiler makes roughly 100,000 calls while compiling the optimizer. With the above figures one can easily derive a rough percentage for the call-push-return overhead in the PL/I compiler--19%. This amounts to roughly 8 seconds out of the total of 40 seconds.

This result of 19%, which differs greatly from earlier estimates of 5%, plus suggestions from several other people, led me to try to get similar figures from a standard benchmark, the "Roach script" which MIT uses to track Multics performance. A special version of pl1_operators_ was made which counted every non-quick call, and the Roach script was run using the special pl1_operators_. (The additional overhead to meter the calls was one AOS instruction per call. In order to keep pl1_operators_ encacheable, only one process was used in the metering run.) The script used 29.979 seconds of virtual CPU time and made 78702 non-quick calls. Using the estimate of 76 microseconds per call gives 5.98 seconds spent in call-push-return for an overhead of 20%, which agrees closely with the earlier figure of 19%. This new figure of 20% shows that the call-push-return overhead is more significant than previously thought.

Recent estimates by knowledgeable people claim that we can reduce the number of instructions executed in pl1_operators_ by call-push-return from 44 instructions to 19 instructions for a 33-50% improvement in the overhead. This would translate into 6 2/3-10% improvement in system performance on the Roach script. It is estimated that this would require about 6 man-months of work. I believe that we have here an opportunity to make a significant performance enhancement to Multics.