TO: Distribution List

FROM: J. F. Gimpel, room 3G-123, BTL, Holmdel, N. J.

SUBJ: A Proposed Outerview of Performance Monitoring in Multics

DATE: April 13, 1966

ABSTRACT

This document supplements an earlier document (B0039) entitled "Software Tools for Monitoring and Tracing in Multics", dated 2/15/66. Whereas the earlier paper dealt with the mechanics of implementing performance monitoring tools (an innerview), this present paper explains how these tools might be used in practice (an outerview). Together, they form an overview of a proposed scheme for monitoring the performance of Multics.

A Proposed Outerview of Performance Monitoring in Multics

by James F. Gimpel

Preface

The Multics repository document BOO39 [1] describes the aims and methods of a proposed Monitoring and Tracing package (MT-package). As such, that paper represents an innerview of performance monitoring. This present paper is a companion to [1]. It speculates how the MT-package might appear to the outside world and therefore represents an outerview. Together these two papers comprise an overview of a proposed scheme for monitoring the performance of Multics.

In some ways the general approach adopted in these papers is novel.* Just why this approach was taken at this time and in this way is treated slightly in [1] and is amplified and expanded in the introduction to this paper.

The body of the paper is intended to be both tutorial and precise. The first to describe what we are talking about and the second to describe what we are not talking about. The first aim is served by an annotated example (section II), the second, by an annotated syntax (section IV). Both properties are essential if feedback is to be obtained and comment is invited.

Whenever the subject of performance monitoring is brought up in conversation, the question of security inevitably comes up. The arguments that are furthered to prevent controlled probing into the system are quite specious as Section III demonstrates.

As is often the case, the ideas contained in this paper are a composite of the thoughts and feelings expressed by other workers on the project which include D. Farber, V. Vyssotsky, C. Jones, J. C. Noll and P. Neumann of BTL and E. L. Glaser, J. Saltzer, D. Widrig and D. Wagner of MIT.

---

*but not completely new. Traces of the same general idea appear in [5], [6] and [7].

Cleverness will be needed and 11th hour decisions will be made; this much is true. But the trickery can be combined early in the game into a comprehensive set of monitoring tools to implement later decisions. This is the general view which is being proposed here.

II. AN EXAMPLE

There is a great deal of data that could be measured and monitored in Multics. The ready list, for example, could be copied at each process exchange in order to obtain a display similar to the login and waiting - queue display at project MAC. Selected contents of the core map could be continually copied for on-line display or off-line perusal. The i/o queues could be copied at each call to a device interface module and either displayed or tested for unusual properties. A histogram of the duration of time spent within various system modules might be interesting. Any of a large variety of statistics such as the average hold-time of a remote console, the average job length, the percentage of system overhead measured in time or in core storage, the percentage of core typically kept impotent waiting for i/o, could all prove valuable within certain contexts. All of these examples have the common property that they can be implemented with the tools described in this paper. Other sorts of possibly more sophisticated monitoring may occur to the reader now, a year from now, or even some time after obtaining the results of other monitoring. We would like to ensure now that such monitoring specified in the future will be possible without major surgery to the existing softwave.

For the purpose of illustration we choose the example of measuring and/or displaying the extent to which a compiler, say the Fortran compiler, is being paged.* We decide that we would like a histogram prepared for each monitored compilation of a Fortran program. The histogram would display the number of times a particular page of the compiler is brought into core versus page number.

Now it is possible for several Fortran compilations to be proceeding nearly simultaneously in the sense that several active processes are using Fortran. We arbitrarily stipulate

*The example would be equally valid for a PL/I compilation, or any other compilation, or indeed any other procedure, by a simple substitution of symbolic names. A simple modification would allow monitoring the paging of data bases as well. The interested reader will probably want to furnish this example for himself.

Table 1

The succession of monitoring modes in the example.

Trigger	Monitoring Mode
	None assumed
Console Request	
	Look for Fortran compilation in any process
Call to Fortran	
	Stop looking for new calls to Fortran; start looking for page-not-in-core faults in the process in which the call was made. Also remain sensitive to a return from Fortran. Monitoring mode does not change with a page-not-in-core fault but some data gathering activity can be invoked.
Return from Fortran	
	Stop looking for page-not-in-core faults and start looking for fresh calls to Fortran. Also write data out.
Call to Fortran	
	Repeat above sequence.
30 min after original request	
	None

the Fortran compiler and gives this number as an argument when arousing File2. In this case, arousing File2 with this argument represents an efficiency as we will see when we examine the second mode of monitoring. In general, the ability to arouse a plan with furnished arguments is a fairly powerful technique. It's implications are far from completely understood. At the moment the syntax allows only arguments in calls out to be parameterized; this is perhaps conservative.*

The process process1, which is scheduled upon return from Fortran, is used merely to convert raw data collected during the compilation into a form more palatable for human consumption.

Plan File2

File2 (see Figure 2) has three requests. The first asks that all page-not-in-core faults be directed to a procedure called tally. The second request asks that tally, as well as the activity distribution routines that call tally, be bolted to core until this plan is suspended. The third request asks that the plan File2 be unregistered after 30 minutes.

The first line of File2 indicates parameters which are plugged at arouse-time. At the moment, only arguments in calls out are regarded capable of being parameterized.

We note also that the activity

```
perform call tally (n)
```

suggests that tally has an argument list of one argument. If the three symbols "(n)" were absent, tally would be called with a standard argument list like alpha in plan File1.

Procedure tally is shown in Figure 4. It is written in PL/I primarily for documentation rather than any other reason; the procedure is quite machine dependent.

III. SECURITY

Inevitably, when the subject of performance monitoring comes up, the problem of security is raised. The issues of security and reliability are important and cannot be lightly dismissed.

*There is, in fact, substantial room for upgrading the entire scheme. For present purposes, the syntax has been kept deliberately simple. There is such a thing as being too ambitious.

where filename indicates the name of a file which contains a plan and the scalar-expressions are interpreted as strings to indicate the name of a registered plan. We have assumed throughout that there will be some standard transformation for subroutine calls to commands as indicated in [3].

Plans

A plan has the following format

```
plan   ::=   [PARAMETERS-line]...request...

PARAMETERS-line   ::=   PARAMETERS identifier...
```

Note: The identifiers which appear in a PARAMETERS-line can appear later in the same plan only in a type-2 activity as indicated below in its syntactic definition.

```
request   ::=   EA-request | FIX-request
```

Note: EA-request is short for EVENT ACTIVITY request.

$$\text{EA-request} \quad ::= \quad \{\{\text{UPON event}\}\ldots \quad [\text{FOR process-class}]\ldots\}\ldots$$
$$\{\text{PERFORM} \left[\begin{Bmatrix}1\\2\end{Bmatrix}\right]\text{activity}\}\ldots$$

Note: The semantic interpretation of the EA-request syntax is probably self-evident.

$$\begin{aligned} \text{event} \quad ::= \quad & \text{CALL-RETURN external-symbol} \mid \\ & \text{FAULT integer} \mid \\ & \text{INTERRUPT integer} \mid \\ & \text{REFERENCE external-symbol} \mid \\ & \begin{bmatrix}\text{AROUSE-TIME +}\\ \text{REGISTER-TIME +}\end{bmatrix} \text{absolute-time} \mid \\ & \begin{Bmatrix}\text{OPEN}\\ \text{CLOSE}\end{Bmatrix} \text{filename} \end{aligned}$$

The interpretation of these six kinds of events is further described in [1].

```
upon call-return fortran
for all processes
perform call alpha
perform(2) restart process1

upon registertime + 30 min.
perform call unregister ("file 1")
```

Figure 1. The contents of file file1.

```
alpha:    procedure  (parameter - list);
          if (this is a call) then go to alpha1;
          if (this is a return) then go to alpha2;
alpha1:   call suspend ("file1");
          /*find somehow the segment number associated with
          Fortran and stuff it into n */
          call arouse ("file2", n);
          return;
alpha2:   call suspend ("file2");
          call arouse ("file1");
          return;
          end alpha;
```

Figure 3. Procedure alpha written in "pidgeon" PL/I.

REFERENCES

[1]. J. F. Gimpel, "Software tools for monitoring and tracing in Multics" Multics Repository Document B0039; Feb. 15, 1966.

[2]. "IBM Operating System 360 PL/I: Language Specifications", IBM Systems Reference Library, File No. S360-29, Form C28-6571-2, Jan. 1966; p. 11.

[3]. D. Eastwood, G. Schroeder, R. Sobecki, "Overview: Use of Commands in Multics", MSPM, BX.0.00; Jan. 27, 1966

[4]. D. Widrig, "System Metering", Multics Repository Document M0062; March 17, 1966

[5]. W. S. Brown, "Debugging and MULTICS", Multics Repository Document B0021; Nov. 9, 1965, p. 4 (last paragraph).

[6]. D. B. Wagner, "Debugging aids for the Multics System", Multics Repository Document M0052; Nov. 9, 1965, p. 2 (first paragraph).

[7]. J Ridgeway, "Trace and dump for EPL procedure debugging in the 645 simulator environment", MSPM, BE.12.01; April 8, 1966

REFERENCES

[1]. J. F. Gimpel, "Software tools for monitoring and tracing in Multics" Multics Repository Document B0039; Feb. 15, 1966.

[2]. "IBM Operating System 360 PL/I: Language Specifications", IBM Systems Reference Library, File No. S360-29, Form C28-6571-2, Jan. 1966; p. 11.

[3]. D. Eastwood, G. Schroeder, R. Sobecki, "Overview: Use of Commands in Multics", MSPM, BX.0.00; Jan. 27, 1966

[4]. D. Widrig, "System Metering", Multics Repository Document M0062; March 17, 1966

[5]. W. S. Brown, "Debugging and MULTICS", Multics Repository Document B0021; Nov. 9, 1965, p. 4 (last paragraph).

[6]. D. B. Wagner, "Debugging aids for the Multics System", Multics Repository Document M0052; Nov. 9, 1965, p. 2 (first paragraph).

[7]. J Ridgeway, "Trace and dump for EPL procedure debugging in the 645 simulator environment", MSPM, BE.12.01; April 8, 1966