THIS REPORT SUPERSEDES TR # 13. THIS WILL APPEAR IN IEEE IN SUMMER 1975 EDITION.

5/23/75

Computer System Monitors*

by

Gary J. Nutt
Department of Computer Science
University of Colorado
Boulder, Colorado 80302

January, 1975

* This work was supported by the National Science Foundation under Grant Numbers GJ-660 and GJ-42251. A preliminary version of this report appeared under the title "Computer System Monitoring Techniques", University of Colorado, Department of Computer Science Technical Report No. CU-CS-013-73.

INTRODUCTION

The most important questions to be answered before attempting to monitor a machine are determining what to measure and why the measurement should be taken. There is no general answer to these questions although a comprehensive set of considerations has been discussed elsewhere [3,16]. The following example indicates some of the considerations involved. Suppose one is interested in tuning a medium scale system which utilizes virtual memory to support a batch multiprogramming strategy. The nature of the job load is a major factor in determining system performance; the mix may be monopolized by I/O-bound jobs which utilize very little processor time. In this case, the bottleneck might be the mass storage system or the peripheral devices. Resource utilization of the peripheral devices may indicate bottlenecks at that point; high mass storage utilization may not be attributable only to the I/O operations, but may be significantly influenced by the virtual memory replacement policy. Processor utilization in this system is also an insufficient measure for most purposes, since the overhead time for spooling, multiprogramming, and virtual memory may be unknown. A more useful measurement for operating system policy studies would quantify processor utilization for the user as well as for each function of interest in the operating system. From this example, one can see that the variety of evaluation objectives

and computer systems causes the determination of what and why to be largely a heuristic problem.

The purpose of this paper, then, is not to answer the two critical questions stated above, nor to survey the entire field of computer measurement but to provide an introduction to the considerations involved in how to measure computer system performance once the what and why have been determined. In the following subsections, some basic terminology is first introduced and basic motivational factors are discussed. The next section describes several quantities that deserve consideration in choosing a method of monitoring. Finally, the alternative monitoring techniques of software monitoring, hardware monitoring, and hybrid monitoring are described.

Basic Terminology

The following basic definitions of often-used terms are given as a basis for further discussion of monitoring techniques. These definitions are descriptive rather than rigorous, since precision usually depends on the particular project at hand.

The throughput rate of a computer system is the average number of task completions per unit time. This description of throughput suffers from ambiguity in the definition of a task completion. Some possible definitions establish the time required for task completion as the time required for a task to fulfill central processor execution. Alternatively, task time may be the sum of queue times for input and output as well as the time during which the task was active, i.e., resident in the primary memory. In an interactive system, a task can be considered to be a single response to a request made by an interactive user.

A definition of turnaround time is also vague, due to the problems mentioned in conjunction with determining throughput. The average turnaround or response time is the average amount of time required for the system to complete a task. Effective turnaround to the batch user not only includes machine residency, but also includes the time that the user waits for an operator to place his deck of cards into a reader and the time span from print completion until the operator returns a listing. The

effective response time for an interactive system corresponds to the system response time. In many cases, the significant portion of turnaround is due to the time required for human action.

A resource in a computer system will be considered to be any portion of the hardware or software system which must be allocated to a job in order for the job to continue execution. For example, a hardware resource might be a tape drive, a portion of the primary memory, or the central processor. A software resource might be a reentrant compiler, a supervisor call, or an input spooling routine. Resource utilization is usually defined as the fraction of real time during which the given resource has been allocated to a job. At other times (e.g. in considering resources that always belong to the operating system such as the area of primary memory containing system tables) the resource utilization is determined as the fraction of real time during which the resource is actually in use. Thus resource utilization is often determined by the level of analysis of the system. In most cases, the applicable definition is obvious.

The competition for allocatable system resources can create performance bottlenecks in which some of the competing jobs are prevented from progressing due to the previous allocation of a desired resource. The most common instance of such a bottleneck results when several

jobs are frequently requesting the central processor; if the set of jobs requesting the CPU is almost always nonempty, then the progression of activity for these jobs is impeded and the central processor is said to be a bottleneck. Other frequent bottlenecks in a system are the primary memory and the disk system in a multiprogrammed computer.

The job load (or work load) of a computer system is the set of all programs, data, and commands that are submitted to the computer system for subsequent execution. A major factor in determining any measure of performance of the system is the corresponding job load of the system. There are several ways to characterize a batch or time sharing workload, and some of these techniques are discussed in the next section.

A system profile is a graphic representation of the activity of various resources of the system, [4,8,9]. In Figure 1, a typical system profile of the activity of a CPU and a channel are shown as they might have been obtained through a monitoring session. The figure indicates the total activity of each unit, the fraction of time in which exactly one is busy, the fraction of time that both units are busy, and the fraction of time that neither is busy. Cockrum and Crockett present a thorough analysis of the use of system profiles for predicting the effect of changing the operating rates of either of the constituents, [8].

Motivation for Monitoring

Some specific motivational factors for monitoring a computer system are: to tune software, to improve the system/user interface, to determine the character of a job load, to investigate resource utilization, or to provide data for system models. Each of these topics are briefly discussed below.

Many installations provide a simple program to be run in conjunction with the target program in a multiprogrammed (or multiprocessor) system which monitors the execution of the target code [5,6,38,47]. The simplest form of program monitor gathers a distribution of instruction fetch addresses. These data indicate heavily used portions of code and can be used to optimize the user program by expending more effort on writing efficient code, perhaps at the assembly language level, for the heavily used portions. Monitoring the execution of the user code can also lead to the discovery of inefficient data structures which result in extra amounts of memory access or bit manipulation.

More general studies of user programs have been made to understand the structure of the program. High level language measurements can be used to understand full statement execution frequencies and the resulting time required for the statement execution [23,36]. Measurements at the statement level often require modification to the user program.

The implementation of virtual memory systems on production systems has led to the need for monitoring a program to determine its demand on the paging (or segmentation) algorithms (e.g. see reference [26]). The programmer can drastically influence the number of page faults by the locality of his program.

Much effort has been expended measuring the job load on a computer system [12, 19, 42]. These studies are generally motivated by the desire to understand how the magnitude and character of the job load will grow. Measurement of users often results in some characterization of the load that can be used for selection studies or that can be used to drive models of systems for new designs or system tuning. Job load studies also include measurements of the user of a given system to determine the relative ease of use of that system, [11]. For example, how much time does it take a naive user to learn how to use a batch system compared with the time required to learn how to use an interactive system? Given that the users know how to use the respective systems, what amount of time is required to solve certain problems, i.e., construct a program that computes the desired result? What are the attitudes of the users toward the respective systems? Questions of this nature are considered by Gold and others [15].

Characterization of the user into certain classes is not always the approach used in a performance evaluation study, especially if the load is being analyzed to drive a model of a prospective system. It is possible to use a real job load by establishing a trace of the actual load placed on a given system. This trace can then be subjected directly to the model, without actually measuring the user and his characteristics [7, 30, 40].

The most popular reason for monitoring the activity of a computer system is to measure resource utilization in order to determine bottlenecks to throughput and/or turnaround time. Countless examples of this motivation for entering a monitoring study exist in the literature, (e.g. see the bibliographies in the papers by Bell [2]; Drummond, [9]; Estrin et al [11]; Goodman, [16]; Lucas [24]; Miller, [27]). It might also be desirable to monitor resource utilization to investigate the relative merits of implementing a system function in either hardware or software (e.g. see reference [17]).

A final motivation for measurement is the desire to obtain data to verify the representativeness of a model. Suppose that an abstract model of a given system is constructed, and the validity of the model must be insured. Measures can be taken on the target system and on the model, as they are both operating on the same job load. The resulting measurements should compare favorably for a valid model, [30].

FACTORS IN CHOOSING A TECHNIQUE

Some Trade-Offs

The choice of a method of monitoring system performance must consider a number of trade-offs or alternatives. The first trade-off involves the sensing of the status of the system. One may choose to record every occurrence of a selected number of events (called full trace monitoring), or alternatively, the monitor may sample the status of the system at selected intervals of time. The full trace method of monitoring may result in an exorbitant amount of data, with the accompanying problems of recording and analysis. However, the complete reaction of the machine is known. Sampling the system status can reduce the volume of data, but introduces new problems concerned with the number of samples taken over the observed period of time and the randomness with which the samples are taken. One must ensure that a sufficient number of samples have been taken in which the randomness of sampling is not synchronized with the occurrence of the variables being measured in the host system.

Indications of system activity can sometimes be measured by monitoring software events or hardware events. At other times, neither measure by itself may be entirely sufficient. Consider a study of drum input/output activity, where it is desired to know the distribution of access times. The drum access time is initiated by a "START I/O" command (see figure 2). At some later time the I/O is initiated on the drum, causing (hardware) latency time to be the limiting factor toward the goal of accomplishing the access.

At time t_2, the drum is positioned under the read/write head and the actual drum I/O transfer starts and lasts until time t_3. At time

t_4, the central processor is interrupted to inform the requesting process that the I/O operation is complete. By monitoring the software, the "START I/O" and "I/O COMPLETE" signals can be recorded and the time interval, t_4-t_0, is the time required for the drum I/O. If the resulting access time is deemed to be unusually high, the software events will not tell the analyst why the time span is long; it may be due to poor scheduling, the time interval given by t_1-t_0; or it may be the result of latency time, t_2-t_1. A hardware-event measurement will provide the time interval t_2-t_1, but does not indicate the scheduling time required. Hence, in this example, measuring either the software exclusively or the hardware exclusively may lead to no insight into a bottleneck to drum I/O.

The choice of the level of monitoring must be made when planning a measurement study. Will the goals of the study be satisfied using macroscopic analysis such as turnaround times, peripheral equipment utilization, etc.? Or is it necessary to drop the level of evaluation to the microscopic level of the previous example. We shall generally differentiate between macroscopic and microscopic monitoring by noting the time units involved in the measurements. Although there is no clear-cut boundary, macroscopic analysis can be thought of as analysis in which the time units involved are seconds, minutes, or hours. Microscopic analysis is characterized by measuring performance in terms of milliseconds or microseconds. The total time for monitoring at the macroscopic level might be a matter of hours and for the microscopic level, a matter of seconds.

The data collected by a performance monitor must be analyzed at some time, either in real time as it is collected (sometimes called

continuous monitoring) or later by a normal batch program. Many of the earlier monitors were developed to collect data and save it on an auxiliary medium such as magnetic tape for later batch processing, (e.g. see references [21] and [45]). However, at least some online data reduction and presentation has been shown to be quite useful, (e.g. see references [5] and [18], with more detailed reports being produced in the offline mode.

Methodology

The level of sophistication of monitoring techniques varies from simple human observation of the machine to complex hardware or software monitors. In the following subsection, five different monitoring methods are introduced.

The simplest form of system monitoring is human observation of an active machine. Bell points out some "audio and visual clues" to understanding where inefficiencies might exist [2]. He notes that a piece of unit record equipment emits a sound whenever it processes a record. In a multiprogrammed system suffering from severe disk contention, a synchronized printing pattern from the several printers may result when printing output files. This is caused by the movement of the disk head from track to track, providing output data first to one printer and then another. When one of the line printers becomes idle, the other printers increase their tempo, indicating faster operation.

The system console can also provide an indicator of system activity. The IBM System/360 has a WAIT light on the console that is turned on whenever the central processor is in the wait state. Observing patterns and relative times that the WAIT light is turned on can help to show the analyst when central processor utilization is low. In some cases, the programmer can detect heavily used portions of code by observing the address indicator lights on the console.

Peripheral equipment use can also provide clues to inefficiency. Whenever a magnetic tape is backspaced and re-read, inefficiency may exist either due to error reads or writes or to inefficient programs employing the backspace command. It is always possible that there is

a good reason for what appears to be inefficiency. The human observation is only useful in giving the analyst clues to inefficiency, but does not always point directly to a performance bug.

A step up in the level of sophistication of monitoring, is the use of a system accounting log to indicate system inefficiency. A system log provides some high level information on the performance of the machine in the form of job occupancy times, processor charge times, unit record equipment utilization, etc. The monitoring is performed as a part of the accounting system and is available to the analyst for some high level evaluation. Accounting log analysis will be discussed in the software monitor section.

In the absence of performance data from human observation or the accounting log, data may be obtained by one of three general methods: Hardware monitoring, software monitoring, or through the use of a hybrid of hardware and software monitoring. A hardware monitor is both logically and physically distinct from the host computer (i.e. the computer being measured). A software monitor is an integral part of the software of the host system and is both logically and physically a part of the host system. A hybrid monitor usually employs a physically separate set of hardware components that are activated by the software of the host system; thus it is logically a part of the host system, but physically distinct. Detailed discussions of each type of monitor are presented subsequently.

Some Potential Pitfalls

There are three general problems to be faced when obtaining performance data (excluding the question of what to measure): the introduction of artifact; provision of a suitable clock; and determining what can be measured with the given tool.

Artifact is the effect on the target system that is caused by the introduction of a monitoring device. The introduction of artifact into a system can be very serious or may not affect the system to any noticeable degree. It is critical that the amount of disturbance caused by the measurement device be well understood. This is not always the case in monitoring. For example, it has been mentioned that the system accounting log can be used to obtain system measurements. Is it reasonable, then, to relate accounting log overhead to measurement, or is that a natural function of the system? Suppose that it is agreed that the accounting log maintenance is a standard system function; many systems are modified so that the accounting log contains more information than accounting information, e.g., queue dwell times, resource utilization statistics, etc. Does this modification introduce measurement artifact or simply more system overhead? These questions may only be answered for a particular installation with its particular set of circumstances.

In other situations, the artifact is well-defined and taken into consideration. For example, the CDC 6000 series has an architecture which provides for ten peripheral processors in addition to the central processor. It is possible to use one of the peripheral processors as a hardware monitor [45]. The amount of disturbance due to the removal of one peripheral processor from the system can be accounted

for if the particular system does not suffer from peripheral processor saturation. Under the saturation case, the effect of removing a peripheral processor from the set of available resources is critical and difficult to quantify.

Software monitors are implemented as a portion of the host system. The measure of artifact is often determined to be the amount of central processor overhead to execute the monitor code. The effect of memory contention introduced by the monitor in accessing tables or using an output controller, channel, and device is often not considered.

Gathering system data which is to be analyzed requires the existence of a clock which the monitor can readily access. In the case of a hardware monitor the clock may be a component of the monitor itself, or the host system clock may be used. Software monitors must rely on the system provided clock, thus it can introduce artifact if it accesses the clock frequently enough to prevent normal system functions from reading the clock.

Internal clocks are maintained by the system supervisor. An external device provides signals which are interpreted as clock "ticks" and cause some memory cell to be incremented for each tick. In some systems, the supervisor is interrupted, (with a high priority interrupt) whenever it is time to increment the clock memory location. In other systems, the supervisory code is written so that the supervisor must check the external device and effect an incrementation as a function of the amount of code executed in the program. If another program reads the external device to maintain its own memory word clock, there is the danger of interfering with the supervisor when it needs to access the external device.

Clock resolution for a measurement device is critical to the interpretation of the data gathered from the host system [44]. If resolution is too large, i.e., the time between ticks is excessive, the clock may not be appropriate for the measurement of events that take place within a relatively short amount of time. The amount of time separating event occurrences will be lost. If a clock with insufficient resolution is used to time the duration of certain events, the durations may be measured as requiring zero amount of time. If clock resolution is too fine, other problems may plague the analyst. Any memory cell of n bits which is incremented at each tick of the clock will cycle through the entire range of (integer) times in 2^n time units. Hence, if the time space of interest in the measurement is much greater than 2^n time units, the meaning of the clock reading will be ambiguous.

A final general problem in choosing a monitor is to decide what measurements are possible in a given system. Suppose that analysis of memory references is to be made in a certain (short) time span. Then the monitoring device must be capable of recording the occurrence of an event in the same amount of time as the memory cycle time (or faster). All of this collected data must somehow be saved for further analysis, hence a clever method of storing a mass of data must be present in the monitor. A case in which the design of the monitor has influenced the system design is discussed by Ferrari, [13].

Data Presentation

The variety of methods for data presentation is limited only by the imagination of the analyst. Among the most popular means of describing the monitored data is the histogram and the system profile. Histograms can be used to illustrate a distribution of turn-around or response time, resource utilization, etc. much more effectively than by providing a mean and variance. An example of a system profile was previously given to illustrate resource utilization and overlap. Recently, Kolence and Kiviat have introduced a radial representation of data that is useful in describing a system profile, [22, 31]. A Kiviat figure is a circle with unit radius and various axis emanating from the center of the circle to the circumference. Each axis represents a fraction of the total time during which the corresponding condition associated with the axis is true. The system profile shown in Figure 1 can be drawn as a Kiviat figure as shown in Figure 3a, where the axis at "12 o'clock" corresponds to "cpu active", the axis at "4 o'clock" represents "channel active", and the axis at "8 o'clock" represents the condition "cpu and channel active". If the cpu and channel had been 100% busy, i.e. full overlap, the resulting Kiviat figure would be as shown in Figure 3b. Although the figures have not withstood the test of time, they seem to be quite promising, [25, 41].

PURE SOFTWARE MONITORING METHODS

During the late sixties software monitoring was the most popular of the techniques discussed in this paper. This probably resulted from the popularity of performance evaluation in multiprogrammed systems by people with strong software backgrounds (i.e. system programmers) and the trend toward more complex software systems in general. Only more recently have hardware and hybrid monitoring techniques been widely used. Each software monitor is highly dependent upon the particular operating system of the host machine; the structure of the operating system and its tables dictate the structure of the software monitor. Although there is no general form of software monitor, one can classify the techniques as follows: System accounting logs, interrupt-driven monitors, and sampling monitors. Many studies incorporate more than one technique, (e.g. see references [5, 31, 38]).

System Accounting Logs

A system accounting log is provided with most operating systems, particularly if the job cost is a function of the amount of various resources used by the job. The simplest system log provides, as a normal operating system activity, messages that indicate the time of day for which various activities occur and the job name to which each activity is related. The content of individual messages indicates the time at which a job becomes active, i.e., is loaded into primary memory; when the job becomes inactive, i.e., releases the primary memory space; the amount of central processor time charged to the job; the amount of I/O time charged to the job; the amount of primary memory used within each job step; and the initiation and termination time of day for each job step. Typically, peripheral equipment assignments

are also provided by the system accounting log. Given this minimum amount of information about each job that is processed by the system, a substantial understanding of the activity can be obtained by analyzing this data by another program. The analysis program can be designed to read the system accounting log, synthesizing system activity by the occurrence of the individual messages. The resulting analysis will provide statistics of chargeable central processor activity, primary memory utilization, I/O resource utilization, and peripheral equipment assignment time. Data is also available to determine the distribution of turnaround time (with respect to primary memory activity), the distribution of the level of multiprogramming, and an indication of the average throughput rate for the analysis period. With some ingenuity, the simplified system accounting log can be used to obtain even more data about the operation of the machine (for example, see references [33, 41, 48]).

Since some system logs include even more information than the minimum amount discussed above, correspondingly more performance information can be obtained from the standard system log. For example, if calls to language processors cause a message to be written to the log, their respective frequency of use can be obtained. It is sometimes possible to derive compile-to-execution time ratios for these systems, if compiler call and termination messages both appear in the system log.

As the system log provides a medium for understanding the system activity, it also allows one to study the characteristics of the job load. The various jobs that enter the machine are recorded as well as a few of their characteristics. Hunt et al have used the system accounting log to derive job classes representing the various

types of users in a university environment [19].

The existence of a capability to provide a system log has led some investigators to take advantage of this capability to obtain more measures about system activity. Stanley describes a Job Accounting System that has been written for a real-time operating system for an IBM 360 [44]. The Job Accounting System gathers the usual accounting statistics in addition to certain desirable data for performance evaluation studies. Additional data that is gathered includes I/O frequencies, system I/O wait time, time for which the central processor is in the wait state but at least one I/O device is operational, system idle time, and system task central processor time. The resulting analysis provides a more detailed report of system activity. Stanley reports that the inclusion of the extra monitoring activity reduces processing capabilities by about one percent (in terms of the amount of central processor and memory used).

System accounting logs are a rather limited medium for obtaining performance measures, their primary asset being that accounting log analysis causes very little disruption to the normal activity of the computing center. It is generally not possible to obtain sample data using the method, but a macroscopic trace of system events is provided.

Interrupt-Intercept Monitors

An interrupt-driven operating system is one in which each invocation of a portion of the operating system is caused by an interrupt, a trap, or a supervisor call. For example, cpu scheduling takes place when a time-slice interrupt or a request for I/O occurs; another por-

tion of the I/O routines is activated with an I/O complete interrupt; etc. Since these interruption points occur whenever the state of system resources is changed, they are a logical time to carry out monitoring activity. The scheme, perhaps best discussed by Keefe, [20], is to intercept each interrupt as it occurs by modifying the interrupt branch table to cause control to be passed directly to a monitoring routine rather than to the operating system interrupt handler. The monitoring routine can then analyze the cause of interruption and inspect system tables before passing the interrupt on to the operating system.

A simple example illustrating the interrupt-intercept monitor is provided by Saltzer and Gintell [38]. Suppose there is a time slice interrupt for central processor multiplexing. Then at the end of a time slice, an interrupt occurs which is intercepted by the software monitor. The monitor then reads and records the contents of the instruction counter register, and returns control to the central processor scheduling portion of the supervisor. The resulting data can be sorted to provide a distribution of addresses and their frequency of access by the control unit.

Using the interrupt-intercept method a trace of system activity is easily obtained, as explained by Cantrell and Ellison, [6]. In the GECOS operating system, major events include central processor allocation, servicing various interrupts, processor mode changes (slave-to-master and vice versa), memory compaction, swapping, etc. Each event occurrence results in a message being written to a circular list and eventually being written on an output device. The content of each message is dependent upon the kind of event that has been sensed. It is possible to mask out each type of event if the event is of no interest to the current analysis. The capability can

be used as a very powerful monitoring device for system activity or any activity of a subset of the total system. This trace mode is used only during monitoring periods, thus the artifact introduced by the built-in monitor affects overall performance only during analysis periods. In the same operating system, Campbell and Heffner ". . . have found that the normal traces cause a system speed degradation of only a few percent." [5, p. 907].

The interrupt-intercept method of system monitoring has the distinct advantage of allowing measurements to be taken as an integral part of the system rather than as an intruder. It is difficult to assess the exact amount of artifact in terms of extra tables or counters included in the system introduced by such an approach. If the system is to be monitored at all, this inclusion of tools as a part of the operating system is the cleanest approach to software monitoring. The time artifact for the method of monitoring can be exorbitant. However, the amount of artifact is relatively easy to obtain. It is also required that the software monitoring program run at a very high priority, to prevent other interrupts from deactivating the monitor.

Sampling Monitors

Sampling monitors are the easiest type of monitor to implement. They require the least amount of operating system modification, but may introduce a significant amount of processor time, memory space, and I/O artifact. The concept is simple, and the implementation of a simple sampling monitor can be accomplished by "non-system programmers," although a thorough knowledge of the system structure is required.

A sampling software monitor may be written as a normal user program for a multiprogramming system. The monitor is activated at (possibly random) periods of time to read the contents of the operating system tables. The operating system interface must include provisions for monitor activation and, if there is a memory access protection mechanism, allow the monitoring program to read the various system tables. The particular tables that the monitor accesses are a function of what tables are available and what tables contain information of interest. The selection of inter-sample periods is critical in that it must not be synchronized with the occurrence of events which are being measured by the monitor. A discussion of a commercially-available sampling software monitor can be found in reference [21], where a set of monitors for various versions of IBM SYSTEM/360 operating systems are described. Another example of a simple sampling monitor is provided by Waite, [47]. In this example, the monitor inspects the user's location counter to obtain a distribution of memory references. The resulting histogram is useful in spotting heavily referenced portions of the program, indicating areas where code optimization can be fruitful.

PURE HARDWARE MONITORS

A pure hardware monitor is a unit that is both physically and logically distinct from the computer system being measured. The interface between the monitor and the device is limited to physical probes used to pass electronic signals from the host computer to the monitoring device. A very simple example is now given to illustrate the concepts involved.

By recording the IBM SYSTEM/360 WAIT light activation, an indication of central processor activity can be obtained. Stang and Southgate describe a simple and inexpensive method for monitoring the WAIT light [43]. A connector is inserted between the bulb and its socket; the connector provides leads that can be connected through various circuitry to a strip chart recorder. Circuitry is included to remove noise and control the rate of operation of the strip chart recorder. The resulting trace represents central processor utilization.

Activity of logical combinations of units can be recorded if each event of interest has a corresponding light on the system console. Connectors may be inserted in the proper sockets, providing leads to the monitoring device. The leads from the console are then routed through logical AND or OR gates before recording, indicating when multiple resources are active.

The Components of a Hardware Monitor

More general hardware monitors employ the same basic technique as discussed above (see Figure 4). The critical item needed for a hardware monitor is the existence of some electronic signal that indicates the occurrence of an event. It is not always possible to monitor console lights, since many events take place that do not have a

corresponding light. However, there is usually a signal on some circuit in the machine. The signal may be of relatively low voltage and the introduction of a monitoring probe can disturb the normal reaction of that circuit. Most internal circuits are designed such that they do not have the power to drive external monitors. To circumvent this problem, high impedance probes can be used in the place of an "alligator clip". A high impedance probe does not draw enough power from the host circuit to cause any adverse reaction. The signal that is observed by the probe is amplified and passed to a signal filter for subsequent processing. The particular probe used for any circuit family must have certain properties in order to accomplish this task. The impedance level, pulse duration, inter-pulse duration, and pulse level all enter into the design of the probe, (see the paper by Noe for a discussion of these considerations, [32]).

Once a signal has been sensed, it is routed to the combinational logic unit of the monitor. This unit, available on most hardware monitors, is provided to allow masking and combination of signals. For example, a probe may be recording processor activity, and another may be recording channel activity. The combinational logic unit can cause any logical combination of those events to be recorded. By connecting the probe leads to a logical AND unit, an event occurrence is recorded whenever both the processor and the channel are active. If the two probes are connected to an exclusive OR unit, sequential activity of the two host components is recorded. This unit may also filter certain signals by ignoring them under a predetermined set of circumstances, [1, 14, 28, 35]. The logic of the combinational unit is frequently determined by a plugboard, although more sophisticated

hardware monitors may use software within the monitor itself to select logical combinations of signals. The outputs from the combinational unit are then routed to a Time and Count unit.

The Time and Count unit contains a set of registers, sometimes consisting of a small memory module, to temporarily record either the amount of time for which some signal exists or the number of times that the signal is sensed. The meaning of the contents of a counter is determined by the location of the probes in the host machine and the status of the combination unit. The process of determining which counters to increment provides the greatest variance in the architecture of different hardware monitors. In the simplest case, a decoder can be used to break a set of related signals corresponding to a channel number or address into a single signal associated with that channel number or address. The single signal is then used to increment a single counter (register) or to begin accumulating a time for which the signal state is constant, (e.g. see references [1, 9]). For example, a 4 x 16 decoder can be used on a 16-bit address to determine the frequency with which 4K pages are referenced. In Figure 5, the most significant four bits of the address are passed through the combination unit to a decoder in the Time and Count unit. A signal on the set of four lines is decoded to activate exactly one of the lines leading to a counter. Decoders with more inputs and outputs can be used to handle smaller page sizes and/or larger addresses.

Fryer, [14], and Murphy, [28], as well as others use an associative (content addressable) memory to aid in event recognition. The basic approach is to save the sensor state (i.e. configuration of a set of signals) in the index field of the associative memory if it has not

previously been entered; if the state has already been saved as an index, the associative memory indicates a "hit" at the memory address containing that state. In either case, a corresponding memory location in a random access memory can then be incremented to save the event count. Other Time and Count units may incorporate comparators for signal comparison and sequencers to detect a predetermined sequence of events as sensed by the monitor; Drummond provides a good explanation of the use of these components, [9].

A hardware monitor may be designed to periodically dump counter contents to a mass storage device after sensing a particular set of signals, or periodically in order to prevent counter overflows. The mass storage recording medium most often employed in a hardware monitor is magnetic tape. Magnetic tape has the advantage of being relatively inexpensive without being too slow to be used for gathering large amounts of data in a short period of time. A third virtue of magnetic tape is that it is machine processable by another computer. The current real-time clock reading is included with each record written to the mass storage medium, to be used for off-line analysis. In order to minimize data loss during the monitor write cycle, either the monitoring process must be suspended or some form of buffering must be used.

Alternatively, the monitor may be used to do online analysis of the accumulated data, to provide the analyst with an immediate indication of the performance of the machine. (Even those monitors that do no online analysis provide a minimal amount of feedback to the analyst via console lights or registers.) This online analysis does not preclude further offline analysis and has been found to be extremely

useful in various performance evaluation studies, [1, 5, 35]. The current trend in hardware monitors is to include this capability.

HYBRID MONITORS

In the previous two sections, pure software and pure hardware monitors have been described. Software monitors have the advantage of being able to relate event occurrences with the stimulus of the event, since the monitor is usually a part of the operating system and is cognizant of the set of processes currently active. Unfortunately, software monitors are not always able to take a needed measure, due to the limitations inherent to the instruction set of the host machine; this form of monitoring can also be criticized for its either undetermined or exorbitant amount of time artifact introduced by the monitor. Hardware monitors are capable of sensing a wide variety of hardware and software events but are limited in their ability in detecting the stimulus for the set of responses. An apparent panacea is the merging of the two techniques to form a hybrid monitor, [11].

The underlying premise of the hybrid monitor is that a hardware monitoring device is not invisible to the operating system, but instead, it is treated as an "intelligent peripheral device" which can be used by a software monitor portion of the operating system, [18]. Nemeth and Rovner, [29], describe a facility of a system that incorporates built-in sensors and event counters that may be allocated to users one-at-a-time (i.e. "serially reuseable") under the control of software. The device has been used to derive an online histogram of subroutine utilization, to investigate virtual memory performance of a single program, and other similar tasks. The allocation of the hardware portion of the monitor has provided the ability to establish the cause-effect relationship. The paper by Stevens, [45], describes another application of a hybrid monitor approach using the existing

hardware of a system, (this approach has also been employed by Control Data in their "PARTNER" monitor, [49]). In this case, the Control Data 6000 series computer system architecture includes at least one central processor and ten peripheral processors. A peripheral processor (PP) can be viewed as a very intelligent channel with a private memory; the PP is an allocatable resource that communicates with peripheral devices and with the central memory. The approach is to allocate the PP along with a tape drive to a user job (which may or may not be dormant); the PP then executes a program from its local memory to sample the state of the system via the operating system tables and the peripheral equipment status indicators.

More recent hybrid monitors have been built in which the hardware monitor portion takes on the character of the monitors described in the previous section, [1, 10, 17, 18, 37, 39, 46]. These monitors are characterized by the use of a minicomputer to control the functions of the hardware monitor under the direction of data obtained via the hardware probes and additional information passed to the monitor through programmed data transfers. The system/monitor interface technique used by Hughes and Cronshaw, [18], and by Ruud, [37], best illustrate the trend toward hybrid monitoring and so that technique will be described.

Figure 6 is a block diagram of a hardware monitoring device integrated into the host system as a peripheral device. Without the connection to a data channel, the system is no different from previously described hardware monitors. This additional data path (e.g. a direct memory access channel) can be used to pass information, including interrupts, both ways between the monitor and the host system.

In order to detect a software event in the host system, the operating system portion of the monitor can pass an interrupt to the external monitor, causing subsequent data transfers identifying appropriate causes of the condition, or the external portion of the monitor can be stimulated by the status of the usual hardware probes. The hardware monitor may also interrupt the host machine to cause software status information to be passed to the monitor via the channel. In this manner, the most significant liability of the pure hardware monitoring technique can be overcome, i.e., causal relationships can be defined. In addition, the power of a hardware monitor is added to the software monitoring technique, allowing for more extensive measurements to be taken. The problem of software monitor artifact still exists, but the effect can be minimized.

The existence of the minicomputer in the monitor also allows for much broader use of the unit than could be expected of the earlier hardware monitors. The minicomputer can be programmed to reconfigure the combinational logic unit upon detecting conditions either via the probes or direct host machine control. The operation of the Time and Count unit can also grow more sophisticated by employing the minicomputer to detect pertinent conditions for recording. The minicomputer also offers the ability to produce reasonably comprehensive online reports and could also be used to produce even more detailed summary reports whenever it is not being used to control monitoring.

From this discussion of hybrid monitoring techniques, it is apparent that sophisticated monitors of the immediate future will be of this class. Perhaps the most important consideration in assessing this technique is the cost of employing such a monitor to obtain

performance evaluation data. If the particular circumstances surrounding the need for measurements require only that the average turnaround time be determined, it would be unreasonable to place such a large investment in a monitoring device when an accounting log analysis would accomplish the goal. Thus we close our discussion of monitoring techniques as we opened it: The questions of what to measure and why the measurement should be taken must be first considered before choosing a monitoring technique. Only after some criteria have been chosen can one consider monitoring techniques.

BIBLIOGRAPHY

1. Aschenbrenner, R. A., Amiot, L., and Natarajan, N. K., "The Neurotron Monitor System", Proceedings of the FJCC, Vol. 39, pp. 31-37, 1971.

2. Bell, T. E., "Performance Determination - The Selection of Tools, If Any", Proceedings of the NCC, Vol. 42, pp. 31-38, 1973.

3. Bell, T. E., Boehm, B. W., and Watson, R. A., "Framework and Initial Phases for Computer Performance Improvement", Proceedings of the FJCC, Vol. 41, Part II, pp. 1141-1154, 1972.

4. Bonner, A. J., "Using System Monitor Output to Improve Performance", IBM Systems Journal, Vol. 8, No. 4, pp. 290-298, 1969.

5. Campbell, D. J., and Heffner, W. J., "Measurement and Analysis of Large Operating Systems during System Development", Proceedings of the FJCC, Vol. 33, pp. 903-914, 1968.

6. Cantrell, H. N., and Ellison, A. L., "Multiprogramming System Performance Measurement and Analysis", Proceedings of the SJCC, Vol. 32, pp. 213-221, 1968.

7. Cheng, P. S., "Trace-Driven System Modeling", IBM Systems Journal, Vol. 8, No. 4, pp. 289-290, 1969.

8. Cockrum, J. C., and Crockett, E. D., "Interpreting Results of a Hardware Systems Monitor", Proceedings of the SJCC, Vol. 38, pp. 23-28, 1971.

9. Drummond, M. E., Evaluation and Measurement Techniques for Digital Computer Systems, Prentice-Hall, 1973.

10. Estrin, G., Hopkins, D., Coggan, R., and Crockers, S. D., "SNUPER Computer--A Computer in Instrumentation Automation", Proceedings of the SJCC, Vol. 30, pp. 645-656, 1967.

11. Estrin, G., Muntz, R. R., and Uzgalis, "Modeling, Measurement and Computer Power", Proceedings of the SJCC, Vol. 40, pp. 725-738, 1972.

12. Ferrari, D., "Workload Characterization and Selection in Computer Performance Measurement", IEEE Computer, Vol. 5, No. 4, pp. 18-24, July/August 1972.

13. Ferrari, D., "Architecture and Instrumentation in a Modular Interactive System", IEEE Computer, Vol. 6, No. 11, pp. 25-29, Nov., 1973.

14. Fryer, R. E., "The Memory Bus Monitor -- A New Device for Developing Real-Time Systems", Proceedings of the NCC, Vol. 43, pp. 75-79, 1972.

15. Gold, M. M., "Time-Sharing and Batch Processing: An Experimental Comparison of their Values in a Problem-Solving Situation", Communications of the ACM, Vol. 12, No. 5, pp. 249-259, May 1969.

16. Goodman, A. F., "Measurement of Computer Systems", Proceedings of the FJCC, Vol. 41, Part II, pp. 669-680, 1972.

17. Hakozaki, K., Yamamoto, M., Ono, T., Ohno, N., and Umemura, M., "Design and Evaluation System for Computer Architecture", Proceedings of the NCC, Vol. 42, pp. 81-86, 1973.

18. Hughes, J., and Cronshaw, D., "On Using a Hardware Monitor as an Intelligent Peripheral", ACM SIGMETRICS Performance Evaluation Review, Vol. 2, No. 4, pp. 3-19, Dec., 1973.

19. Hunt, E. B., Diehr, G., and Garnatz, D., "Who are the Users: An Analysis of Computer Use in a University Computer Center", Proceedings of the SJCC, Vol. 38, pp. 231-238, 1971.

20. Keefe, D. D., "Hierarchical Control Programs for System Evaluation", IBM Systems Journal, Vol. 7, No. 2, pp. 123-133, 1968.

21. Kolence, K. W., "A Software View of Measurement Tools", Datamation, Vol. 17, No. 1, pp. 32-38, January, 1971.

22. Kolence, K. W., and Kiviat, P. J., "Software Unit Profiles and Kiviat Figures", ACM SIGMETRICS Performance Evaluation Review, Vol. 3, No. 1, pp. 34-39, March 1973.

23. Knuth, D. E., "An Empirical Study of Fortran Programs", Software -- Practice and Experience, Vol. 1, 1971.

24. Lucas, H. C., "Performance Evaluation and Monitoring", Computing Surveys, Vol. 3, No. 3, pp. 79-92, September 1971.

25. Merrill, H. E. B., "A Technique for Comparative Analysis of Kiviat Graphs", ACM SIGMETRICS Performance Evaluation Review, Vol. 3, No. 1, pp. 34-39, March, 1974.

26. Millbrandt, W. W., and Rodriguez-Rosell, J., "An Interactive Program Evaluation", Proceedings of the NCC, Vol. 43, pp. 153-158, 1974.

27. Miller, E. F., "Bibliography on Techniques of Computer Performance Analysis", IEEE Computer, Vol. 5, No. 5, pp. 39-47, September, 1972.

28. Murphy, R. W., "The System Logic and Usage Recorder", Proceedings of the FJCC, Vol. 35, pp. 219-229, 1969.

29. Nemeth, A. G., and Rovner, P. D., "User Program Measurement in a Time-Shared Environment", Communications of the ACM, Vol. 14, No. 10, pp. 661-666, October, 1971.

30. Noe, J. D., and Nutt, G. J., "Validation of a Trace-Driven CDC 6400 Simulation", Proceedings of the SJCC, Vol. 40, pp. 749-757, 1972.

31. Noe, J. D., and Runstein, N. W., "Continuous Computer Performance Monitoring", Unpublished paper, July, 1973.

32. Noe, J. D., "Acquiring and Using a Hardware Monitor", Datamation, Vol. 20, No. 4, pp. 89-95, April, 1974.

33. Nutt, G. J., "Computer System Resource Requirements of Novice Programming Students", University of Colorado, Department of Computer Science Technical Report No. CU-CS-039-74, 1974, (to appear in Software--Practice and Experience).

34. Pinkerton, T. B., "Performance Monitoring in a Time-Sharing System", Communications of the ACM, Vol. 12, No. 11, pp. 608-617, Nov., 1969.

35. Roek, D. J., and Emerson, W. C., "A Hardware Instrumentation Approach to Evaluation of Large Scale Systems", Proceedings of the ACM National Conference, pp. 351-367, 1969.

36. Russell, E. C., and Estrin, G., "Measurement Based Automatic Analysis of FORTRAN Programs", Proceedings of the SJCC, Vol. 34, pp. 723-732, 1969.

37. Ruud, R. J., "The CPM-X -- A Systems Approach to Performance Measurement", Proceedings of the FJCC, Vol. 41, Part II, pp. 949-957, 1972.

38. Saltzer, J. H., and Gintell, J. W., "The Instrumentation of Multics", Communications of the ACM, Vol. 13, No. 8, pp. 495-500, August, 1970.

39. Shemer, J. E. and Robertson, J. B., "Instrumentation of Time-Shared Systems", IEEE Computer, Vol. 5, No. 4, pp. 39-48, July/August, 1972.

40. Sherman, S., Baskett, F., and Browne, J. G., "Trace-Driven Modeling and Analysis of CPU Scheduling in a Multiprogramming System", Communications of the ACM, Vol. 15, No. 12, pp. 1063-1069, December, 1972.

41. Snyder, R., "A Quantitative Study of the Addition of Extended Core Storage", ACM SIGMETRICS Performance Evaluation Review, Vol. 3, No. 1, pp. 10-33, March, 1974.

42. Sreenivasan, K. and Kleinman, A. J., "On the Construction of a Representative Synthetic Workload", Communication of the ACM, Vol. 17, No. 3, pp. 127-133, March, 1974.

43. Stang, H., and Southgate, P., "Performance Evaluation of Third Generation Computing Systems", Datamation, Vol. 15, pp. 181-190, November, 1969.

44. Stanley, W. I., "Measurement of System Operational Statistics", IBM Systems Journal, Vol. 8, No. 4, pp. 299-308, 1969.

45. Stevens, D. F., "System Evaluation of the Control Data 6600", Proceedings of the IFIP Congress, pp. 034-38, 1968.

46. Svobodova, L., "Online System Performance Measurements with Software and Hybrid Monitors", ACM Fourth Symposium on Operating System Principles, pp. 45-53, October, 1973.

47. Waite, W. M., "A Sampling Monitor for Applications Programs", Software -- Practice and Experience, Vol. 3, pp. 75-79, 1973.

48. Watson, R. A., "Computer Performance Analysis: Applications of Accounting Data", Rand Report No. R-573-NASA-PR, May, 1971.

49. --, "6400/6500/6600 PARTNER Installation Manual and Operating Guide", Control Data Corporation.