SPY—A program to monitor OS/360

by R. SEDGEWICK, R. STONE, and J. W. McDONALD

Western Electric Engineering Research Center
Princeton, New Jersey

INTRODUCTION

It is generally agreed that one of the major problems facing the manufacturers of large scale computer systems today is the problem of measuring the performance of a computer in conjunction with the operating system which drives it. This problem is under consideration for operating systems currently in the design and implementation stages through studies on: (i) the establishment of reasonable criteria under which performance can be measured; (ii) means of actually making the measurements; and (iii) ways of using the information obtained to optimize the performance of the system under the established criteria for a particular user's environment.

For existing operating systems the problem is somewhat different, especially in cases where it is not feasible to make any changes to the operating system itself. First, extracting information from the system is a non-trivial problem under the constraint that the system cannot be modified. Second, the information obtained is little more than academic unless it reflects inefficiencies in the system caused by the environment of user jobs being processed, as optimization can only be effected by restructuring this environment or by setting parameters to direct the system to operate according to the environment.

A related problem that has arisen with the advent of time-sharing and multiprogramming systems is that of monitoring system execution. This is an important area of research for two reasons: First, one characteristic of many existing operating systems is that they just cannot function without a fair amount of operator intervention, and the operator must know exactly how the resources of the system are being utilized in order to carry out his job effectively. Second, situations often arise in a multiprogramming or time-sharing environment in which it is not altogether clear exactly what action is being taken by the system. The ability to observe the performance of such a system in operation is of importance to systems programmers and managers to enable them to more effectively "tune" the system and watch the local effects of any changes made. This ability can also be of value to users of a machine—the typical user has no concept of his impact on the operating system, and a certain amount of awareness can enable him to better appreciate how incorrect estimates of his system resource requirements can affect the entire user community.

A considerable amount of work has been done in the general area of system performance measurement in the past, and a wide variety of approaches to the problem are represented.[4,5,11] In general, however, they can be roughly categorized into three classes: theoretical studies (simulations and statistical work); hardware monitoring devices; and software data gathering techniques.

The theoretical studies[2,8,10] generally focus on the much broader problem of improvement of operating system design so that maximum efficiency can be provided under very carefully thought out criteria. The process is to make use of thorough statistical analyses in conjunction with generalized computer system models to develop a coherent theory which clearly defines (and generally suggests solutions to) basic problems in operating system design related to overall system performance. This work is invaluable to the system designer, and is a fundamental step in the development of future computer systems, but it is in many cases not directly applicable to the problems associated with existing systems, especially when the point of view is taken that the internal mechanism of the system must be viewed essentially as a fixed entity.

Hardware measurement[3,6] is accomplished through the use of an independent set of instruments which have the capability of sensing, decoding, and recording selected electronic signals in the system being measured. This approach to the problem has the decided advantage that the measurement device can be used indiscriminately with little effect on normal system operations—a voluminous amount of data can be obtained on the operation of the system in its natural state. The danger, of course, exists that a good deal of extraneous data could be obtained—the amount of useful data is dependent on the sophistication of the interface to the host computer and the amount of flexibility in the measurement device itself. (An example of a very flexible device can be found in the SNUPER COMPUTER[3] project, where an auxiliary computer, which can be programmed to accept or reject available data, is used as a measurement device.) The main disadvantage in the use of hardware techniques is that it is often difficult to correlate the data being gathered with the software being run on the machine or the software representing the internals of the operating system. The data gathered often shows very clearly how effectively the hardware is being utilized, but aside from reconfiguring the hardware it is in general not clear what steps could be taken to improve upon the observed results.

Software measurement techniques[9,12] generally consist of data gathering programs which run while a system is in normal operation, and reduction procedures which extract useful measurements from the large volume of data thus obtained. The differences in implementation can be found in the types of events monitored, the rate at which they are monitored, and the algorithms used in the data reduction process. There are many different schools of thought in all of these areas—one obvious reason is that measurement of an operating system is quite dependent on the structure of that system. Not only are the data gathering techniques a direct function of the system being studied, but also significant events in one system are quite different from significant events in another. The principal difficulty with software measurement is that the measurement procedures themselves utilize some percentage of the resources of the system—every effort must be made to minimize this percentage.

When considering software measurement techniques, one must make a clear distinction between measuring the operation of a system from within and observing it from the outside. One feature common to most existing system measurement programs is that they are being developed by people who have the luxury of being able to work with the internals of the system being measured. This allows them to facilitate somewhat the process of extracting information from the system, and to provide very accurate measurements of relevant system events. This type of observation from within can be done at many levels—the most desirable method is to provide for insertion of "hooks" in the system early in the design process, and allow for a workable interface between the measurement procedures and the operating system, so that the system can be made responsive to the measurements obtained. (An example of such a system is the XDS Sigma 5/7 BTM system.[7]) However, the point must again be emphasized that in many cases the operating system must be viewed as an unalterable product of the manufacturer: Measurement must be done by merely looking at the internals; and optimization can only be effected by rearranging the environment in which the system operates.

There is far less work in the literature on the subject of dynamically monitoring system execution. Although some capabilities in this area are necessarily provided in most operating systems, a major obstacle is that output is normally done on a typewriter-like console—the state of a multiprogramming or time-sharing system changes far too quickly for such a device. A graphic display is a much more appropriate device for dynamic monitoring for at least two reasons: First, *all* changes in the system can be recorded, allowing the observer to choose those he may consider significant as he watches the system in operation. Secondly, interactions within the system can be easily observed because all information is presented simultaneously—the effect of an activity in any part of the system can be seen in other sections, as it develops. This type of global view of the dynamics of the system would not be possible on anything other than a graphic display. An excellent example of work in this area is the GDM system at Project MAC,[1] which provides dynamic displays of a very flexible format allowing observation of the operation of the MULTICS time-sharing system.

The subject of this discussion, the SPY project, represents one approach to this dual problem of measuring and monitoring operating systems for an existing multiprogramming system—a version of IBM's operating system, OS/360. A system was developed which presents information concerning the jobs being processed by the operating system, the direct access devices connected to the computer, and the state of the System/360 CPU. Data is in general gathered through software techniques, but a facility for rudimentary hardware monitoring is also employed. The information is displayed dynamically and in real time on a graphic display unit and selectively saved on paper tape for later analysis. No modification whatsoever to OS/360 is involved, and a negligible amount of overhead is incurred, so that the system can be used to analyze and observe the operation of OS/360 in day-to-day operations.

Concepts and definition of terms

The version of OS/360 to be analyzed was MVT (Multiprogramming with a Variable number of Tasks), and any further discussion of our implementation of a monitor would be impossible without the use of some of the terminology and concepts related to OS/360, MVT, and the System/360 hardware. The intent of this section is to provide a rudimentary explanation of the most basic terms.

IBM defines *multiprogramming* as "a general term that expresses use of a computing system to fulfill two or more different requirements concurrently."[14] In general "requirements" are defined in terms of *tasks*, where a task is the smallest independent unit of work for the processor. In multiprogramming, the effect is as though all tasks in the machine at one time were running asynchronously. Tasks can be *system tasks* (readers, writers, schedulers, etc.) or *user tasks*. The users of the system generally submit their work to the system in terms of *jobs*, the individual parts of which are broken off as tasks by the system.

The *computer resources* are those parts of the hardware that can be allocated to individual tasks. The resources about which we will be concerned will be core storage, the central processing unit, and disk (direct access) storage—all of which will be referred to together in this paper as the *resources*.

A *volume* for the purposes of this discussion will be a single disk pack and a *unit* will be the disk drive upon which the volumes can be individually mounted.

The *wait light* is one of five lights on the console of the 360/50 which are meant to indicate the "state" of the machine. In normal operations, when it is off, the CPU is in the process of executing an instruction, and when it is on, the CPU is waiting, generally for some I/O activity.

Hardware configuration

The specific system being analyzed was running on a IBM System 360 Model 50 with 512K bytes of core with an eight drive 2314 direct access storage facility, plus various other peripheral equipment. During development, a 1024K byte 2361 LCS unit and a second 2314 with four drives were added. The monitor system was implemented on this machine and on a PDP-9 with 8K of core equipped with a Graphic-II Display Unit. Communications between the two machines was done via a Parallel Data Adapter (PDA) connected to a 2701 Data Adapter Unit on the 360 and to a specially designed interface on the PDP-9. In addition, the PDP-9 has the hardware capability of monitoring the state of the wait light on the S/360.

It is fully realized that this is a somewhat peculiar hardware configuration, and some effort is being expended to modify the system to allow it to be run on more standard configurations. The only equipment absolutely essential is an S/360 Model 30 or higher and some suitable output device—the above equipment was chosen for use simply because of its availability.

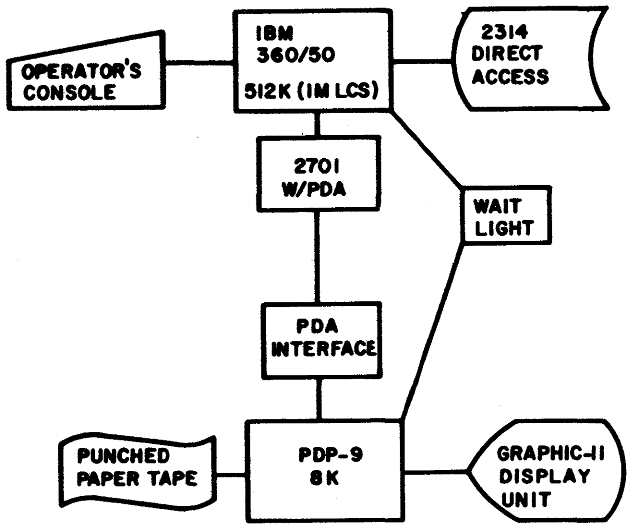


Figure 1—Hardware configuration

Software structure

The final system consists basically of two software packages: one in the 360/50, which takes samples on selected information within OS/360, accumulates statistics, and sends information to the PDP-9 at selected intervals; and one in the PDP-9, which receives the information, independently monitors the wait light, formats all of the information for display, and produces a hard copy record of the important statistics.

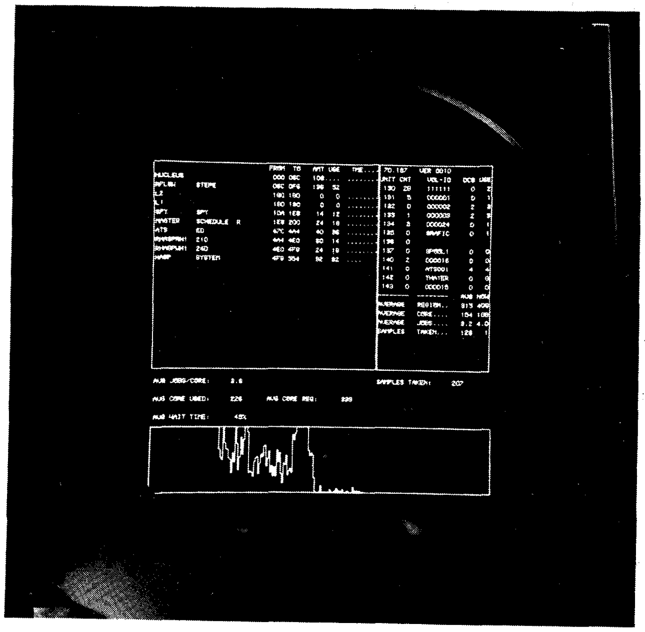
The emphasis in implementation of the programs on the 360 was in modularity, and the final system consists of a set of assembly language subroutines. The modules comprising the OS/360 interface portion of the program were debugged in PL/I, so that use could be made of the powerful I/O and error correcting facilities of the higher-level language. Working modules were then translated into PL/360 to gain the advantages of high speed and low storage requirements implicit in assembly language level programs. This method resulted in the production of efficient code quickly, as the advantages of both levels of programming were made use of, and there was little effort in conversion due to the structural similarity of the languages. The transmission and central control modules of the system were written and debugged in assembly language, simply because the tasks to be performed required low-level interfaces to the system.

All of the PDP-9 software for SPY was written and debugged in assembly language on the PDP-9, chiefly because of the lack of any workable higher-level language, and because of the necessity for low-level interfaces to the special purpose I/O devices (the PDA and the wait light).

GENERAL DESCRIPTION OF INFORMATION RETURNED

The principal output of the SPY system is a dynamically changing graphic display consisting of a wide variety of information relating to the operation of the 360/50 and OS/360. The display is divided into four sections (see Figure 2): one containing specific information concerning the jobs being processed; one showing the utilization of the direct access devices (on which the system is heavily dependent); one containing running averages of selected summary information (the information used in system measurement); and one containing a continually recycling graph showing the percentage of CPU utilization. All of the information is updated dynamically as the events it reflects occur, under the constraints outlined in the section below on timing and sampling.

Also output is a punched paper tape containing the data reflected in the summary statistics as well as some data retained from the samples of the wait light.



```
                                   FROM  TO   ANT  USE   TME...  70.187   VER 0010
NUCLEUS                            000  0BC   108....   ......  UNIT CNT  VOL-ID   DCB USE
HPLAN      STEME                   0BC  0F8   138  52           130  29   111111    0  2
L2                                 180  180     0   0  ......   131   5   000001    0  1
L1                                 180  180     0   0  ......   132   0   000002    2  3
SPY        SPY                     10A  1E8    14  12  ......   133   1   000003    2  3
MASTER     SCHEDULE  R             1E8  200    24  16  ......   134   3   000024    0  1
ATS        ED                      47C  444    40  36  ......   135   0   WAFIC     0  1
HASPSPWN1  210                     444  4E0    80  14  ......   136   0
BSHASPWN1  240                     4E0  4F8    24  19  ......   137   0   SPOOL1    0  0
HASP       SYSTEM                  4F8  554    92  82  ....     140   2   000016    0  0
                                                                141   0   ATS001    4  4
                                                                142   0   THAYER    0  0
                                                                143   0   000015    0  0
                                                                -------   -------   AVG NOW
                                                                AVERAGE   REGION..  815 400
                                                                AVERAGE   CORE....  154 108
                                                                AVERAGE   JOBS....  3.2 4.0
                                                                SAMPLES   TAKEN...  128   1

AVG JOBS/CORE:     3.6                                          SAMPLES TAKEN:    207

AVG CORE USED:     226        AVG CORE REQ:     298

AVG WAIT TIME:     45%
```

Figure 2—Graphical output from SPY

Identification information is also included, so that a large amount of data can be collected over large periods of time and analyzed statistically to aid in drawing some general conclusions on overall system performance.

Timing and sampling

Most of the activity in the SPY system is regulated by a set of three basic timed intervals. In the S/360, these are maintained with the use of the standard task timing facility (STIMER), which allows the system to operate by taking control of the CPU only for a very short time at the end of each interval and relinquishing it during the interval. It is recognized that this practice of relying on the operating system to do the timing may be somewhat unsafe, since irregularities in the timing algorithms of the operating system could lead to irregularities in the data gathered. However, the PDP-9 utilizes a hardware clock for its timing, so that it is able to ensure accurate timing and program synchronization—the STIMER facility is relied upon solely to relinquish control for real time intervals and has proven adequate for this purpose.

So that the system could be made useful as both a measurement and a monitoring tool, two intervals were needed for timings during the execution of SPY. First, a "*transmission interval*," which represents the time between transmissions to the PDP-9, and hence the time between changes in the graphical display. A second interval needed is the "*sampling interval*" or time between the accumulation of samples on the S/360.

A lower bound on the choosing of a transmission interval was set by the ability of the eye to follow changes in the information presented, and an upper bound resulted from the fact that selection of long intervals resulted in the missing of changes in the system. A reasonable compromise between these two limits appeared to be ten seconds.

The selection of a sampling interval was governed by the need to have a statistically significant number of samples, while not spending so much time sampling that performance might be degraded on smaller or slower systems. The choice of one second resulted in an acceptable number of samples and an extremely low overhead.

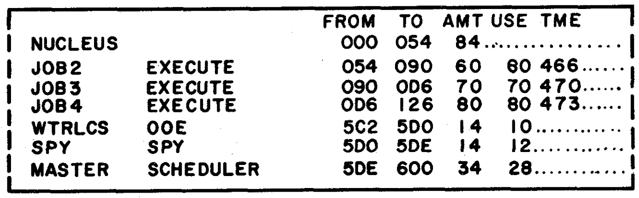
Sampling of the wait light in the PDP-9 represents measurement and monitoring of the S/360 hardware rather than the OS/360 software, so that different criteria are used in the timing. The most significant point is that it was considered necessary to sample the state of the wait light as often as possible—this is done every 1/60 of a second, which is the granularity of the standard clock on the PDP-9. An intermediate average is computed for display purposes every 5/6 of a second, and a final average retained before the graph is recycled. The aim was to produce a dynamically changing graph which moves fast enough so as not to seem frustratingly slow to the observer, and slow enough so that the data displayed is in some way significant.

In addition, it was necessary to define a third interval, a "*statistics gathering interval,*" at the end of which the measurement data is output on hard copy for later analysis. The length of this interval is dictated by the amount of data that needs to be saved, and the rate at which it is being accumulated. In the current implementation of the system, a value of two hundred seconds is used. (This value was chosen for convenience only, as it represents the amount of time it takes the wait light graph to complete one cycle.)

Scheduled jobs and tasks

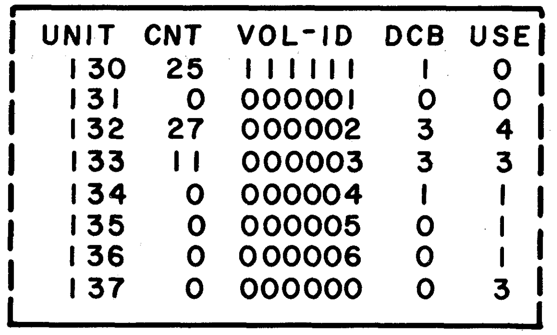
The information given in this section of the display allows monitoring of the various user jobs and system tasks that have been selected for execution by the operating system. There is one entry for each such task, and for each entry, the following information is given:

(i) An identification of the entry (the job name, step name, and procedure name).
(ii) "FROM" and "TO" addresses indicating the locations in main core assigned the program by the operating system.
(iii) The amount of core requested by the program (and the amount given it by the operating system—this number is equal to the difference of the "FROM" and "TO" addresses).
(iv) The amount of core actually being used by the program (in general very different from the amount requested).



```
                          FROM  TO  AMT USE TME
NUCLEUS                   000  054  84...............
JOB2      EXECUTE         054  090  60   60 466......
JOB3      EXECUTE         090  0D6  70   70 470......
JOB4      EXECUTE         0D6  126  80   80 473......
WTRLCS    00E             5C2  5D0  14   10..........
SPY       SPY             5D0  5DE  14   12..........
MASTER    SCHEDULER       5DE  600  34   28..........
```

Figure 3—Typical job and task information



```
UNIT  CNT  VOL-ID  DCB  USE
 130   25  111111    1    0
 131    0  000001    0    0
 132   27  000002    3    4
 133   11  000003    3    3
 134    0  000004    1    1
 135    0  000005    0    1
 136    0  000006    0    1
 137    0  000000    0    3
```

Figure 4—Typical direct access device information

(v) The time remaining in the time interval allotted the program step by the operating system. Changes in this number reflect the relative CPU utilization of each job.

This information presents a true "picture" of the condition of the machine in relation to the jobs being processed—allowing continuous monitoring of the treatment of the various jobs by the operating system, and the use (or misuse) of the system by the various jobs. Figure 3 shows a "snapshot" of this section of the display during normal system operations.

Direct access device utilization

This area of the display contains information concerned with the direct access devices attached to the system. For each direct access unit, five pieces of information are displayed:

(i) The unit number identifying the disk drive.
(ii) The volume name of the disk pack currently mounted on that unit.
(iii) A measure of the number of potential users of the unit. (The number of DD cards pointing to that unit.)
(iv) A measure of the number of users actually using the unit. (The number of open DCBs pointing to the unit.)
(v) A number from 0 to 100 representing the percentage of time the user of the unit changes. (A measure of contention on the unit.)

With this information one is able to monitor and gather statistics on the operation of the direct access devices and their utilization by both the users and the operating system itself. This is very useful because one characteristic of OS/360 is that, in general, effective operation of the system is dependent on effective operation of the direct access devices. Figure 4 shows the information typically displayed in this section. (the CNT field is the number described in (v) above—the other fields are self-explanatory).

Global information

This section of the display contains most of the information used in system measurement. Although it is expected that this section will be expanded in the near future, in the current implementation of SPY it contains summary information concerning the non-system tasks. This information is also useful in monitoring the system, so it is presented on the display as both instantaneous data (the values determined in the sample at the end of the transmission interval), and cumulative data (averages of the values sampled since the beginning of the statistics gathering interval). The following is the data presented:

(i) The average number of jobs in execution (this is a much smaller number than the number of tasks displayed in the section described above—a job can occupy core but not be eligible for execution). This is clearly a very important global measure for a multiprogramming system, as it can be taken to more or less reflect the hardware and software capabilities of the system to run multiple jobs.

(ii) The "average region size" of all of the jobs currently in execution (corresponding to the "amount of core requested" above). This number is an indication of the amount of core given to the users of the system after the space for the operating system and space lost due to fragmentation have been subtracted.

(iii) The average amount of core being used by the jobs in execution (corresponding to the "amount of core used" above). This number, when compared to the average region size, shows the ability of the programs to estimate their core resource requirements.

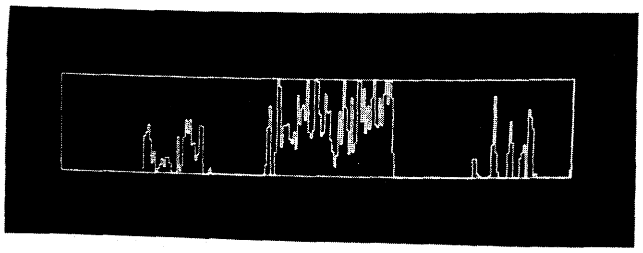


Figure 5—Wait light graph

This information is used in both system monitoring and system measurement to observe the effectiveness of the OS/360 implementation of multiprogramming, the appropriateness of the system parameters that have been selected for the installation, and whether or not the user community is utilizing the system effectively.

CPU utilization (wait light)

The wait light portion of the display consists of a graph on the Graphic-II Display, the X-axis of which is time, and the Y-axis of which is the percentage of time the "wait" light on the 360 is off (i.e., the computer is active).

The result is a dynamic display indicating the degree of activity of the CPU of the S/360 (see Figure 5). Although this information could lead to incorrect conclusions if taken as an absolute measure, this graph has proven very effective, taken in conjunction with the other information displayed, in giving an observer a feel for the load on the CPU at any time.

PROGRAM STRUCTURE

360 program structure

As mentioned above, the 360 program is modular in design and can essentially be divided into three major logical sections: a control program, a transmission package, and an information retrieval package. The control program maintains the three defined sampling intervals and calls the other sections accordingly. The transmission section formats the data for transmission and graphical display and does the actual transmission. The information retrieval package (the OS/360 interface) gathers specified data from the internal workings of the operating system, does the sampling, and puts the data into character format. As required by the design goals, no modification whatsoever is made to OS/360.

This is possible due to the fact that OS/360's data space consists of a series of tables chained together in a complex queue structure.[13] Information about individual tasks in the system is determined by searching the task queue and weeding out the tasks which are active (those tasks which have no "daughter" task on the linked list). Information stored about each task includes its name, core location, core requested, and whether it is a system task. Time information for a task is found from its "mother" task, and actual core utilization is determined by adding the various parts of its region that have not been actually used.

Direct access information is determined by a list of the locations of the status data about the units. This data yields the currently mounted disk pack, whether a mount is in progress, active users, and potential users. In addition, it is possible to trace chains backward to determine the name of the last user, and hence status information is built up concerning the degree of contention.

PDP-9 program structure

The major goal in the development of the PDP-9 program was to produce a clean, readable display, while at the same time acting as a I/O device slaved to the S/360. Functions to be performed were: the reception and display of all the data accumulated by the S/360 programs; the maintenance and display of the wait light graph; and the production of the punched paper tape output. The program gives highest priority to processing related to receiving data from the S/360—at no time does the 360 have to wait for the PDP-9 to accept a transmitted message. If this were allowed to occur, the integrity of the sampling times would be compromised more than necessary.

The main control portion of the program sets up the initial display, initializes the PDA to accept messages from the 360, and sits in a tight loop: awaiting a signal from the PDA handling routine indicating that a complete message has been received; and sampling and maintaining a count on the state of the wait light. At the expiration of every 5/6 second interval, the wait light count is converted into the commands necessary to update the wait light graph, and the main loop reentered. When a message from the 360 arrives, a message decoder determines which information was received and rebuilds the portion of the display which is to be updated. If the message received contained the final average information, an output routine formats and punches out the paper tape hard copy. Upon completion of message processing, the PDA is reinitialized for input and the main loop reentered.

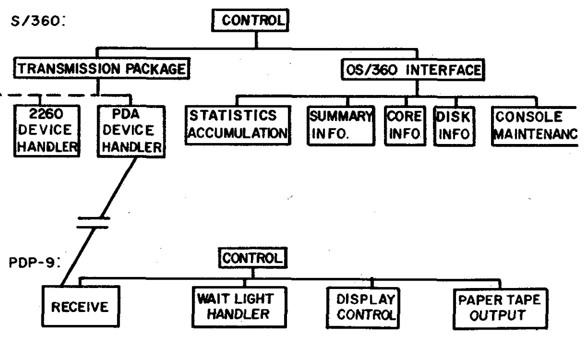


Figure 6—Software structure

COMMUNICATIONS

Introduction to equipment

As mentioned above, the communications link used by the SPY system is a high-speed Parallel Data Adapter interfaced to the IBM 2701. It should be pointed out that transmission at this rate (180,000 bytes/sec) is not crucial to the operation of the system—much slower rates would probably be sufficient. On the other hand, the adapter is almost a hard wire connection between the computers and as such does not require sophisticated low-level programming or extensive error diagnostic routines. The use of the PDA was simply a case of utilizing the most convenient communications device.

Conventions

The communications conventions are rather simple and straightforward—conversation between computers is kept at a minimum and is essentially in one direction with the PDP-9 acting as the receiver and the 360/50 as the transmitter. Acknowledgment of reception is contained and diagnosed within the hardware and the channel of the PDA so that the customary acknowledgment transmission is unnecessary.

The transmission buffers are of a constant length—2048 bytes. The data are preceded by a one byte code describing which type of data format is being sent. All information to be displayed is sent in PDP-9 Graphic-II display code (two 7-bit ASCII characters packed in the low order 14 bits of each 18 bit word) with control characters that determine the physical position on the screen where the data is to be displayed. The "global" information is transmitted in 8 bit ASCII—this is done to make easier the task of producing the punched paper tape. (In future versions of SPY, all character translation will be done in the PDP-9, to reduce overhead on the S/360.)

Transmittal (S/360 program)

The Parallel Data Adapter is a non-supported device under OS/360, and the I/O routines were therefore written at the lowest level allowed users by the system (EXCP—EXecute Channel Program). This has the additional benefit of keeping core requirements at a minimum.

The PDA is OPENed as an output device once for every run of the program. For every message to be transmitted, the appropriate code conversion is performed (input to the transmission package is EBCDIC), a test made to ensure that the previous transmission was successfully completed, and the EXCP macro issued to direct the system to begin transmission.

Reception (PDP-9 program)

Communications on the PDP-9 is done at the hardware I/O level, with the use of a standard routine developed along with the PDA which does the actual READ into PDP-9 core and sets the various hardware flags to notify the 2701 of proper reception. The main PDP-9 program is notified of the reception of a complete message via a simple "event variable."

INPUT-OUTPUT INTERFACES

Graphic-II display

All information presented by the SPY system is displayed on the Graphic-II display unit, a general purpose display driven by the PDP-9. The display processor generates pictures from a "buffer program" in the PDP-9 core which defines displays in terms of point, line, and character information. The PDP-9 program builds and maintains the buffer program and directs the display interface in presenting and regenerating the display.

With the exception of the wait light graph, all of the information presented is in character form, and this information, along with positioning information, is put into the display buffer by the PDP-9 program. The wait light graph is produced by dynamically modifying the display buffer to define small vectors which trace out the percentages measured.

The wait light

The hardware interface to the S/360 wait light consists of an actual hardware connection from the physical wait light socket on the 360 console to the PDP-9, and an interface on the PDP-9 allowing the state of the light to be tested with a single machine language instruction (S360W—Skip on 360 Wait light).

Operator's Console

It was also deemed necessary to include in SPY an interface to the S/360 operator's console, so that a mechanism could be provided to allow modification of some of SPY's internal parameters at execution time. In addition, the facility was included to allow the operator to request any of the information provided by SPY for output on the console.

The I/O function is performed by sending a request for information to the console (WTOR), resulting in a console message to which the operator can reply at any later time. Rather than waiting for this response from the operator, SPY interrogates OS/360 at the end of every one second sampling interval to determine whether the operator has sent a message. When a message is sent, SPY simulates the occurrence of the end of the transmission-interval, but routes the transmission to the operators console instead of the PDP-9.

Planned additions

Now that a working system has been developed and has proved useful, one direction for future research will be to explore the possibilities of incorporating less expensive display devices into the system, for the purpose of allowing SPY to be generated and used on any System 360 for a modest cost. Since most of the information presented is in character form, this is only a matter of acquiring the hardware and programming new "transmission" packages. The first new device to be included in this way will probably be an IBM 2265 Display Station, or some similar device. It should be pointed out that a major drawback in such a system will be the loss of the wait light graph, although ways of providing some type of replacement for this are also being researched. In general, the emphasis in future development will be on complete generality and modularity in relation to I/O devices. Another area that will have to be researched for versions of the system without the PDP-9 will be in statistics gathering and production of hard-copy results. The development of programs on the S/360 which will save data (either on disks or magnetic tape) is under consideration, but still under the basic constraint of minimizing overhead on the S/360.

A separate area of future work will be in researching ways of making maximum utilization of the hardware currently being used, and developing more special purpose hardware to expand the capabilities of the system. As of this writing, this has already been begun in the implementation of a high-resolution (down to 40 microseconds) clock in the PDP-9 to allow extremely accurate wait light sampling, and an expansion of the wait light interface to allow other lights on the 360 console to be monitored.

APPLICATIONS OF THE SYSTEM

Analysis of OS/360 operation

Over the past several months, SPY has been used to aid in analyzing the performance of the 360/50 with MVT, and the following conclusions have been reached concerning the operation of this particular computer facility and the impact of SPY upon operations.

The most serious effect of running a highly variable job stream (i.e., jobs with widely differing resource requirements) in a multiprogramming environment on a computer with no virtual memory capabilities is *"core fragmentation."* The core fragmentation problem can be defined as the situation that exists when all of the core resources of the machine are not being utilized because of the fact that the available core is not contiguous. This situation was observed to be most detrimental to system performance.

In addition, two problems were observed in relation to the direct access devices: First, contention was found to be serious on those devices needed by all the users (i.e., the output spool device). Secondly, a problem was observed in the apparent non-reenterability of the device allocation routine in cases where a new volume is required.

In the area of user impact on the operation of the system, the most harmful thing observed was overestimation of the core resource—ranging from 50 percent to 150 percent. The difference between the amount estimated and the amount used is a totally wasted productive resource. In addition, a similar problem was noticed in the use of the direct access devices—users would have the system allocate devices for their use, and then never use them—another wasted resource.

The general conclusion reached was that resources were being underutilized—not only due to possible system inefficiencies (the operating system generated for this particular machine possibly was not completely tailored to the workload being studied), but also due to some lack of understanding on the part of the user community of the effects (and requirements) of multiprogramming.

Viewing multiprogramming

A motion picture film has been made from the output display of SPY showing a carefully selected job stream. This film has become an extremely valuable tutorial aid in demonstrating multiprogramming to the unsophisticated. To the more knowledgeable, it presents a real look at the actual occurrence of problems in a running computer, and the reasons that cause such a system to fall below the ideal maximum efficiency.

The film begins with a clear system with only SPY present, and then adds a single program and the system tasks necessary for that program. Slowly, the capability for more programs is added until a three job system is achieved. As jobs move in and out of execution, problems of core fragmentation, contention, unused core, and unused disks are observed.

The technique of using a movie, while not changing the amount of information presented, has allowed that information to be presented to persons outside the immediate environment, and has permitted the display of a controlled selection of jobs with a wide range of characteristics.

CONCLUSION

The SPY system has proven invaluable not only as a means of measuring the performance of a computing system but also as a tutorial aid in the theory of multiprogramming.

The statistics gathering capability, as expected, proved to be vital to the functions of "tuning" a system and validating hardware and software changes. Although the measurements retained for analysis in this first version of the system were rather coarse, they did bring to light some fundamental problems in the use of the operating system, and led to some action being taken towards system optimization.

The monitoring capability proved to be very valuable in giving many observers a true feel for how the operating system works. In addition, the availability of operator information was often useful and showed much potential, depending on the types of jobs being run at an installation, and thus the degree of operator control over jobs entering the system.

The two major design goals were clearly met: no modification whatsoever was made to OS/360 in any of the software development; and it was at all times clear that the effects of SPY upon the operation of the system are negligible. It utilizes less than 1 percent of the CPU time, occupies only 14K of core storage, and uses no secondary storage units.

One extension of the work presented in this paper will be the use of SPY to aid in more serious statistical studies on the operation of computing systems. Now that a means has been developed to extract salient information from the system, a serious developmental effort can be expended towards expanding the system to produce data which can be used in more valid and meaningful statistical tests on system performance.

In conjunction with this work it is hoped that not only can the capabilities provided by the system be generalized, but also the applicability of the system can be extended. It is felt that this project has shown such a system to be an effective tool in any computing environment, and it is hoped that this experience can be expanded upon by the implementation of the system at other S/360 installations.

REFERENCES

1 J M GROCHOW
Real-time graphic display of time-sharing system operating characteristics
AFIPS Conference Proceedings Fall Joint Computer Conference Volume 35 pp 379-386 1969

2 G ESTRIN L KLEINROCK
Measures, models, and measurements for time-shared computer utilities
Proceedings ACM National Meeting pp 85-95 1967

3 G ESTRIN et al
SNUPER COMPUTER—A computer in instrumentation automation
AFIPS Conference Proceedings Spring Joint Computer Conference Volume 30 pp 645-656 1967

4 P CALINGAERT
System performance evaluation: Survey and appraisal
CACM Volume 10 No 1 pp 12-18 January 1967

5 R A ARBUCKLE
Computer analysis and thruput evaluation
Computers and Automation Volume 15 No 1 pp 12-15 January 1966

6 F D SCHULMAN
Hardware measurement device for IBM system/360 time-sharing evaluation
Proceedings ACM National Meeting pp 103-109 1967

7 J E SHEMER D W HEYING
Performance modeling and empirical measurements in a system designed for batch and time-sharing users
AFIPS Conference Proceedings Fall Joint Computer Conference Volume 35 pp 17-26 1969

8 T B PINKERTON
Performance modeling in a time-shared system
CACM Volume 12 No 11 pp 608-610 November 1969

9 H N CONTRELL A L ELLISON
Multiprogramming system performance measurement and analysis
AFIPS Conference Proceedings Spring Joint Computer Conference Volume 32 pp 213-221 1968

10 A L SCHERR
An anlysis of time-shared computer systems
Research Monograph No 36 MIT Press Cambridge Massachusetts 1967

11 T HASTINGS et al
Conversational system performance and measurement
DECUS Proceedings Fall Symposium pp 191-201 1969

12 W R DENISTON
SIPE: a TSS/360 software measurement technique
Procedings 24th National ACM Conference 1969

13 *IBM System/360 Operating System: System control blocks*
IBM Corporation White Plains NY Form C28-6628

14 *IBM System/360 Operating System: Concepts and facilities*
IBM Corporation White Plains NY Form C28-6535 1968