A Method for Analyzing Operating Systems utilizing Network Representation

Akio Komatsu*, Shigeru Motobayashi* and Nobumasa Takahashi*

1. INTRODUCTION

In this paper we will introduce a method for analyzing and evaluating operating systems. This method is applicable not only to operating systems, but also to all other software systems: compilers, user programs, etc. The characteristics of this method are: 1) An operating system is modeled in terms of a set of paths (a path is a chain of nodes representing program modules). 2) A technique like one of those used in OR (Operations Research) is applied to this model in order to analyze and evaluate the performance of the operating system.

Based on this idea a program named OS-Analyzer has been developed. The OS-Analyzer consists of the following three phases: 1) static analysis of the structure of the operating system by simply scanning the source program, 2) dynamic analysis and network modeling of the control transitions among the modules using the data obtained by address-tracing, 3) evaluation and simulation of the program behavior by manipulating the model in conversational mode. Structural and functional features, bottlenecks, and the effect of improving a module can be clarified based on the results of these analyses.

In this paper we will also describe some of the results obtained by application of the OS-Analyzer to MCP (Master Control Program: the resident part of the operating system for HITAC 5020 TSS[1]).

2. PROGRAM MODELS

2.1 Network Modeling

The theory of networks provides a useful model for a variety of physical phenomena in traffic, communication, transportation and activity networks[2]. Activity networks, as exemplified by PERT and CPM, form a

This paper first appeared in Japanese in Joho — Shori (Journal of the Information Processing Society of Japan), Vol. 13, No. 12 (1972), pp. 810 ~ 817.

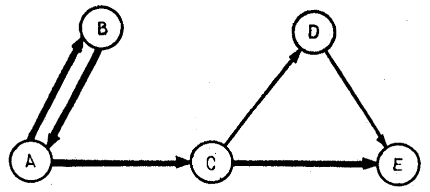
* Central Research Laboratory, Hitachi, Ltd.,
Kokubunji, Tokyo, Japan

special category of networks, and the algebra proposed for analyzing them is limited to only the temporal (i.e., scheduling) relationships among events.

Elmaghraby [2] has generalized the approach of activity networks to networks in which activities and events occur probabilistically and contain more complicated relationships. GERT$^{3)}$ (Graphical Evaluation and Review Technique) is a procedure for analyzing such generalized networks, i.e. stochastic networks composed of EXCLUSIVE-OR, INCLUSIVE-OR and AND nodes, and multiparameter branches.

For the purpose of analyzing and evaluating software systems, we will introduce two types of networks.

At first we will consider a network similar to an ordinary control flow chart of a program, and call this type of network SNF (Static Network Flow). An example of an SNF is shown in Fig. 1. 'Static' implies that an arc in an SNF implicates only the possibility of control transition between two modules (nodes). Since a node represents a program module, a path, i.e. a chain of nodes, represents a history of control transitions among modules. In an operating system, for example, a history of intermodule transitions caused by processing a SVC (Supervisor Call) corresponds to one path, and a course of processing an interruption corresponds to another. In other words, there is one-to-one correspondence between a path and a processing. So a set of paths, each weighted by the frequency of its occurrence, represents a behavior or a workload of the operating system modeled. We call this set of weighted paths DNF (Dynamic Network Flow). It is evident that a DNF includes more information than a stochastic network in the sense that we can derive a corresponding stochastic network from a DNF.



Circles (nodes) represent each program modules, and arrows (arcs) represent intermodule transmission of control. A path is a chain of nodes. (e. g. ABACE)

Fig. 1 Networkflow representation of a program

2.2 Information Needed for Modeling

Now we will explain the kind of information that is necessary for modeling an operating system. The OS-Analyzer refers to the following three tables when constructing an SNF and a DNF.

(1) Division Table

The Division Table is used to divide an operating system into a number of program modules by specifying the address range of each module. An entry in the table consists of a module name and the first and the last addresses of the program module.

(2) Combination Table

The Combination Table is used to increase flexibility in dividing an operating system into its functional units. By making use of this table we can define a new module which consists of a set of modules defined in the Division Table.

(3) Instruction Table

In a table called Instruction Table, a set of instructions and each instruction execution time are defined. Based on this table, the running time of each module or path can be calculated.

3. STRUCTURE OF THE OS-ANALYZER

The conceptual figure of the OS-Analyzer is shown in Figure 2. The OS-Analyzer consists of three phases, each of which is described in this section.

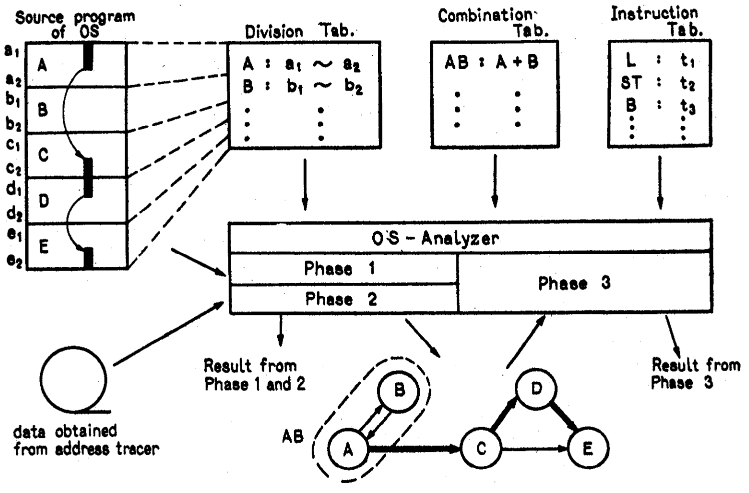


Fig. 2 Conceptual figure of OS-Analyzer

3.1 Static Analysis (Phase 1)

Static analysis is done in Phase 1 using the source program of an operating system and the three tables mentioned in section 2.2. We can obtain the following data through Phase 1.

(1) Information about an SNF

We can clarify static intermodule connections and draw an SNF which represents the structural features of the operating system.

(2) Frequency of each instruction appeared

The frequency of each instruction appeared in the source program of the operating system is counted.

3.2 Dynamic Analysis (Phase 2)

Dynamic analysis is done in Phase 2 using actual data obtained from address tracing, and a DNF is constructed as a more detailed model representing the structural and functional features of an operating system. The output from Phase 2 is listed below.

(1) A sequence of paths

A sequence of paths ordered by their occurrences means a sequence of requests made to the operating system. Based on this sequence, we can analyze the workload on the operating system. From this information we can count the frequency of each path and construct a DNF, i.e. a set of paths weighted by the frequency of their occurrence.

(2) Execution time of each path

The execution time of a path means the processing time needed for the accomplishment of a corresponding function of an operating system.

(3) Average execution time and frequency of use of each node

From these data we can find the program modules which are often used or which need longer elapsed time to process their function.

(4) Intermodule control transition ratio

Together with the average execution time of each node, intermodule control transition ratio can be utilized to construct a stochastic network.

(5) Frequency of occurrence of each instruction

Based on the frequency count of each instruction we can calculate an OS-mix, i.e. an instruction mix in the operating system.

3.3 Evaluation and Simulation (Phase 3)

A DNF obtained in Phase 2 can be regarded as a kind of X-ray photograph of an operating system, reflecting its behavior as well as its structure. In Phase 3, various kinds of evaluation and simulation can be performed as directed using the following commands in conversational mode under HITAC 5020 TSS.

GRAPH: displays the intermodule transition ratio.

TOTAL/PATH/MODULE: displays the statistics on the DNF/paths/modules.

DESCEND: displays the names of all succeeding modules in paths.

SUBPATH: finds the specified chain of modules in paths.

SEARCH: finds modules which have high improvement ratio.

UPDATE: updates the internal data such as weights of paths.

COM: combines a number of modules into one module.

END: terminates the process of Phase 3.

Using these commands we can evaluate and simulate the effect of the changes in workload, execution steps of program modules, etc., on the performance of the operating system. Furthermore we can find bottlenecks or modules of high improvement ratio.

The improvement ratio is defined as follows. Let M_{ik} be an average execution time of the k-th module in the i-th path, P_i (execution time of the i-th path) can be calculated such that $P_i = \overset{k}{\Sigma} M_{ik}$.
Let W_i be the weight, i.e. the frequency of occurrence of the i-th path, E (the expectation of the execution time of a path) can be calculated such that

$$E = \overset{i}{\Sigma} P_i \times W_i / \overset{i}{\Sigma} W_i \ .$$

Now suppose that E changes to E' according as M_{ik} changes to M_{ik}'.
We define the two improvement ratios as follows.

Improvement ratio of a module: $MI = (M_{ik} - M_{ik}') \times 100/M_{ik}$

Improvement ratio of a system: $I = (E - E') \times 100/E$

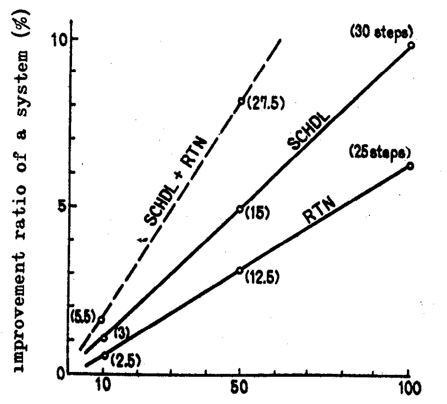
4. RESULTS

We have applied the OS-Analyzer to MCP (Master Control Program) using the data obtained by the address tracing[5]. Some of the results are described in this section.

We divided MCP, which is about 8K words (32 bits/word) in size, into 250 modules. Among these modules, 137 modules appeared in the trace-data. (The others are special modules used exceptionally when abnormal situations occur.) There were 343 arcs in the DNF of MCP. The ratio of the number of arcs to the number of nodes can be considered to be an index to the complexity of program structure.

In the trace data about 90 different paths appeared. The paths associated with the processing of the missing-page faults and drum channel interruptions, occupied 60% of the total frequency of all the paths. Dynamic steps needed for processing missing page faults ranged from 500 to 2000. We can clarify the behavior of an operating system for time-sharing systems utilizing segmentation and paging mechanisms through the detailed analysis of these paths.

Figure 3 shows useful data concerning the improvement ratios defined in 3.3. From this data we can expect that the decrease of 3 dynamic steps (10%) in a module named 'SCHDL' will result in the decrease of about 1% in a average execution time of a path. Furthermore, if we can halve the dynamic steps of the 'SCHDL' and the 'RTN' respectively, the improvement ratio will be over 8%. These statistics are probably open to various interpretations, however, the advantage of utilizing quantitative analysis in improving a program module is not to be taken lightly.



'SCHDL' and 'RTN' represent particular program module names.

Fig. 3 Change of improvement ratio of a system by improving average time of a module

Analyses of other results, such as the ratio of user time to OS time, instruction mix in OS and so on, have been omitted in this paper.

5. CONCLUSION

In this paper we have described a method for analyzing and evaluating operating systems utilizing network representation. The OS-Analyzer has proved to be a useful tool in analyzing and evaluating operating systems through experiments. The OS-Analyzer is a general procedure for systematic analysis of the data obtained by address tracing. Since the results obtained by analyzing MCP (Master Control Program) through the OS-Analyzer were not so different from the ones obtained by another method such as software monitoring, we can conclude that the method presented here is applicable to a wide range of software systems.

ACKNOWLEDGEMENT

We would like to express our appreciation to Messrs. T. Masuda, Y. Yoshizawa and T. Hirosawa for valuable discussions, to our department manager Dr. T. Tagami for his support in this work, and to Dr. S. Shimada for his kind interest in this work.

REFERENCES

1. Motobayashi, S., Masuda, T. and Takahashi, N., "The HITAC 5020 Time Sharing System," Proc. ACM. 24th, Nat. Conf., 1969, pp. 419-430.
2. Elmaghraby, S. E., "An Algebra for the Analysis of Generalized Activity Networks," Management Science, Vol. 10, No. 3, 1964.
3. Pritsker, A. A. B. and Harp, W. W., "GERT: Graphical Evaluation and Review Technique: Part I. Fundamentals," The Journal of Industrial Engineering, Vol. XVII, No. 5, May, 1966.
4. Beizer, B., "Analytical Techniques for the Statistical Evaluation of Program Running Time," FJCC, 1970, pp. 519-524.
5. Masuda, T., Yashizawa, Y., Hirosawa, T. and Takahashi, N., "System Data and its Evaluation of Time Sharing System with Virtual Memory Concept," Proc. IEEE Conf. of Mexico Region, Jan., 1971.